Kizamu: A System For Sculpting Digital Characters

Ronald N. Perry and Sarah F. Frisken
Mitsubishi Electric Research Laboratories

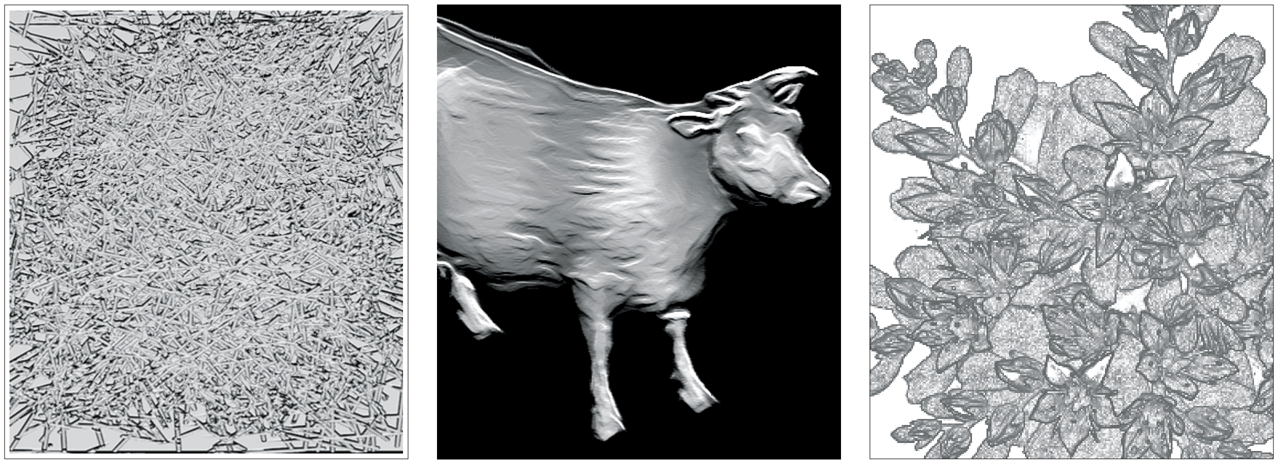


Figure 1. 3D models created with Kizamu ("to carve" in Japanese) and rendered with simple Phong illumination; all texture is purely geometric. On the left – a slab carved with 2000 random chisel strokes (level-12 ADF, 124 MB); in the center – a procedurally sculpted cow; on the right – flowers carved in bas-relief from a slab.

ABSTRACT

This paper presents Kizamu, a computer-based sculpting system for creating digital characters for the entertainment industry. Kizamu incorporates a blend of new algorithms, significant technical advances, and novel user interaction paradigms into a system that is both powerful and unique.

To meet the demands of high-end digital character design, Kizamu addresses three requirements posed to us by a major production studio. First, animators and artists want ***digital clay*** – a medium with the characteristics of real clay and the advantages of being digital. Second, the system should run on standard hardware at interactive rates. Finally, the system must accept and generate standard 3D representations thereby enabling integration into an existing animation production pipeline.

At the heart of the Kizamu system are Adaptively Sampled Distance Fields (ADFs), a volumetric shape representation with the characteristics required for digital clay. In this paper, we describe the system and present the major research advances in ADFs that were required to make Kizamu a reality.

CR Categories: I.3.8 [Computer Graphics]: Applications; I.3.4 Graphics Utilities – Graphics Editors; I.3.5 Computational Geometry and Object Modeling – Object Modeling, I.3.3 Image Generation – Display Algorithms

Keywords: digital sculpting, graphics systems, character design, rendering, ADFs, distance fields, volume modeling, triangulation.

{perry,frisken}@merl.com

ACM SIGGRAPH 2001, 12-17 August 2001, Los Angeles, CA, USA
© 2001 ACM 1-58113-374-X/01/08...$5.00

1. MOTIVATION

Creating convincing digital character models is a major challenge for the entertainment industry. Animation artists generally employ two methods: models are either sculpted from a traditional medium like clay and then digitized, or they are constructed using one of several commercial (or custom) modeling systems such as MAYA or SoftImage.

Because clay can be directly molded and carved to create both smooth surfaces and fine detail, it is the medium of choice for most artists. However, sculpting with clay and then digitizing the clay model (i.e., maquette) has several limitations for digital animation. For example, much detail can be lost in the digitizing process and long-term storage of clay maquettes is difficult.

Alternatively, commercial modeling systems have been developed which use polygons, NURBS, and subdivision surfaces to represent shape. However, all three representations have limitations. Polygon models require many vertices to represent detailed surfaces which makes creation and editing cumbersome. Because NURBS are topologically restricted, they must be patched together to form complex shapes, thus presenting numerous technical and interface challenges [9]. While subdivision surfaces do not suffer from the same topological restrictions as NURBS, controlling shape changes and adding fine detail during editing are difficult. These three representations are all edited by manipulating control vertices, requiring significant skill and patience as well as foresight and careful planning to ensure that models have enough control vertices where detail is desired. For these reasons, object representations used in existing commercial modeling systems do not rival the intuitive and expressive nature of clay.

Because sampled volumes represent object interiors and can be sculpted directly by changing sample values near the tool/object interface, they appear to hold more promise as a data structure for digital clay. However, existing volumetric sculpting systems have large memory requirements and a fixed resolution determined by the volume size, resulting in models with limited detail (e.g., rounded corners). As a result, such systems are not used in the high-end digital animation industry. The limitations of volume representations were recently addressed by the introduction of Adaptively Sampled Distance Fields (ADFs) [12]. In that paper, Frisken *et al.* introduced ADFs, outlined basic algorithms for generating, rendering, and sculpting ADFs, and briefly described a number of applications for ADFs.

We were recently challenged by Industrial Light and Magic (ILM) to explore the use of ADFs for sculpting digital characters; Kizamu is the result. Digital production studios such as ILM have three fundamental requirements for a sculpting system: 1) animators and artists want ***digital clay*** – a medium with the characteristics of real clay (e.g., expressive in its ability to represent both smooth surfaces and very fine detail, intuitive to sculpt, easy to manipulate, and responsive to user input) and the advantages of being digital (e.g., ability to undo, script, duplicate, integrate into digital animation systems, and store permanently), 2) the system should run on standard hardware at interactive rates and 3) the system must fit into an existing production pipeline for animation (e.g., the system must accept standard 3D representations or 3D scanned data from clay maquettes and output models compatible with that pipeline).

During discussions with ILM it became apparent that ADFs have other advantages for the entertainment industry [18]. For example, ADFs integrate volumes and surfaces into a single representation, thereby potentially reducing the number of independent representations in their production system. In addition, ADFs can be combined by simple CSG operations so that individual parts of a character can be modeled separately and later joined together. This feature could increase productivity in character design: libraries of body parts or facial features could be maintained and used to produce new characters quickly and easily.

Given these requirements, we set out to design and build Kizamu based on the ideas presented in [12]. We quickly found that the algorithms and methods for working with ADFs presented in that paper were inadequate for the demands of a production-level system. In this paper, we describe the Kizamu system and present the major research advances in ADFs that were required to build it. The contributions fall into 4 major areas:

1. Innovations in the volumetric sculpting interface that take advantage of the distance field to provide more control to the artist such as using the distance field to orient and position the sculpting tool relative to the surface, using distance values to constrain sculpting tools, and region-based conversion to triangle patches for editing via control vertices

2. Advances in ADF generation and editing such as a new generation algorithm with reduced memory requirements, better memory coherency, and reduced computation, as well as a method for correcting distance values far from the surface during sculpting

3. Several new ADF rendering approaches that take advantage of hardware acceleration in standard PCs, including fast generation of triangles and surface points

4. Methods for inputting and outputting models from the system including an improved technique for generating ADFs from scanned range images and a very fast new algorithm for generating topologically consistent (i.e., orientable and closed) triangle models from ADFs at various levels of detail (LOD)

In Kizamu, we provide artists with an interactive tool for sculpting which: 1) incorporates the ideal interface (digital clay), 2) represents very high resolution shapes, 3) has a relatively small memory footprint, 4) runs on standard hardware, and 5) provides input and output methods for easy integration into existing animation pipelines. To our knowledge, no other system provides this unique blend of features.

While Kizamu was designed to meet the demands of a high-end production studio like ILM, the system has exciting potential in other areas of the entertainment industry such as character design for games and for virtual reality. The ability to generate LOD models with low polygon counts, as required by these applications, enables easy integration of ADF-based characters into existing polygon engines.

2. BACKGROUND

2.1 ADFs

A distance field is a scalar field that specifies the minimum distance to the surface of a shape. When the distance field is signed, the sign can be used to distinguish between the inside and outside of the shape. ADFs adaptively sample the shape's distance field and store the sampled distance values in a spatial hierarchy for efficient processing. Distances at arbitrary points in the field can then be reconstructed from the sampled values and used for processing such as rendering and sculpting. The use of adaptive sampling permits high sampling rates in regions of fine detail and low sampling rates where the distance field varies smoothly, allowing very high accuracy without excessive memory requirements.

Frisken *et al.* discussed a basic sculpting procedure for ADFs and outlined algorithms for generating and rendering ADFs . Like [12], Kizamu uses octree-based ADFs with trilinear distance and gradient reconstruction functions. However, when building Kizamu, we found the algorithms presented in [12] to be inadequate for a production-level system. Specifically, for generation, the proposed bottom-up algorithm requires too much memory and too many distance computations while the top-down algorithm requires expensive octree neighbor searches and unnecessary repeated recomputation of distance values. Both approaches exhibit poor memory coherence for cells and distance values. These memory and processing limitations place practical restrictions on the maximum ADF level that can be generated with those algorithms (see Section 5.1). For rendering, while the ray casting algorithm is sufficiently fast for small local updates on a desktop Pentium system, it is too slow for local updates on low-end systems and woefully inadequate for global view changes such as rotating the object. Finally, in [12], there is no consideration of user interfaces for digital sculpting, no method for conversion to standard representations, no method for correcting distance values away from the surface, no hardware acceleration, and only discrete steps are supported during editing.

2.2 Related Work

There are several commercial systems for creating character models such as MAYA, SoftImage, 3DStudioMax, FormZ, and Houdini. In these systems, polygons, NURBS, and subdivision surfaces are constructed by the artist or generated from 3D scanned data. The surface models are edited by moving control vertices, a process that can be tedious and time consuming. To make this manipulation more intuitive, most of these systems allow the user to interact with groups of control vertices using a sculpting metaphor. For example, in MAYA Artisan, NURBS models are modified via a brush-based interface that manipulates groups of control vertices. Operations for pushing, pulling, smoothing, and erasing the surface are supported where the brush tool affects control vertices in a region of diminishing influence around the tool's center, resulting in a softening of the sculpted shape. The amount of detail in the sculpted surface depends on the number of control vertices in the region of the sculpting tool – finer control requires finer subdivision of the NURBS surface, resulting in a less responsive system and a larger model. The mesh subdivision is user controlled and is preset – it does not adapt to tool selection or the detail of the sculpted surface. Achieving fine

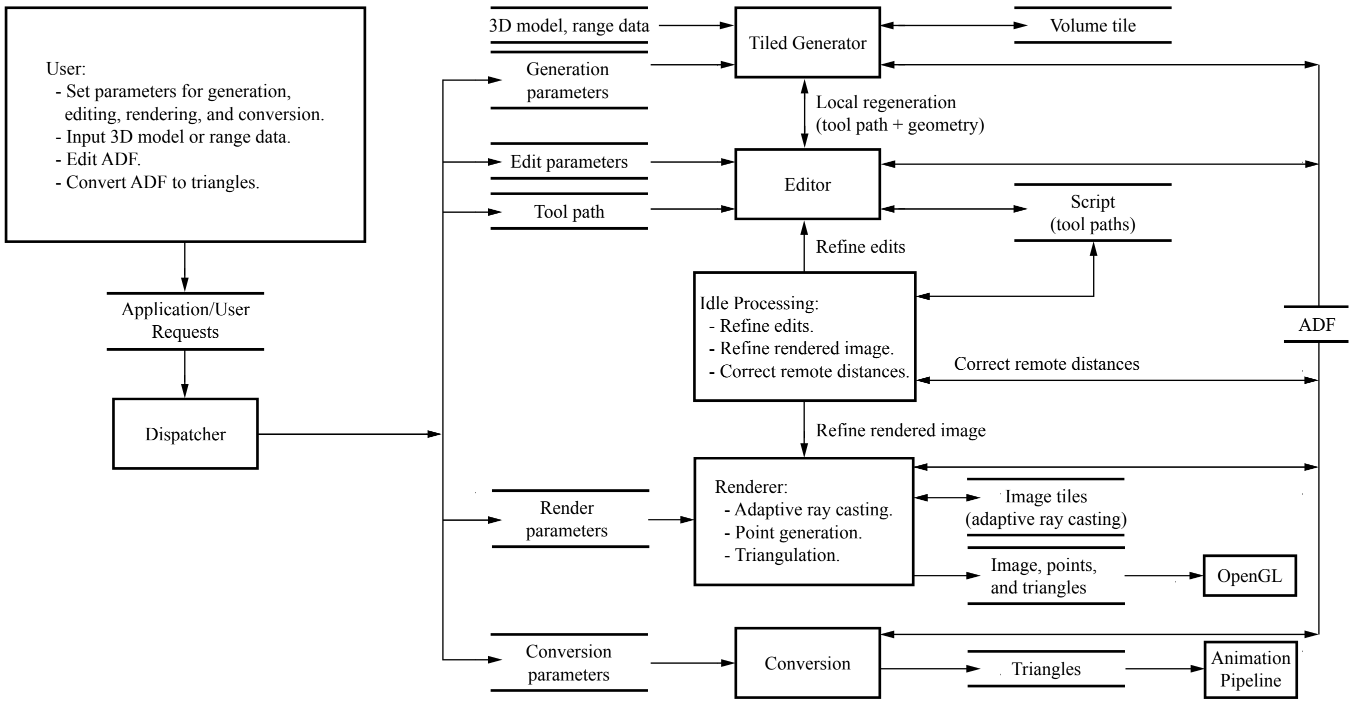


Figure 2. A dataflow diagram of the Kizamu system with input from standard 3D forms and 3D scanned data, ADF generation, ADF editing, ADF rendering, idle-time processing, and conversion of ADFs to triangle models. In this diagram, a box represents a process while text between lines represents data.

detail in desired regions without excessive subdivision of the surface in other regions requires significant foresight and planning. SoftImage provides a sculpting interface, called “Meta-Clay”, that appears to be based on metaballs technology [28]. Meta-Clay is a density-based object representation for modeling organic, sculpted objects. This representation produces blobby shapes and does not represent edges, corners, and fine detail.

FreeForm is a commercial system for sculpting volumetric models. The system includes a 3 degree-of-freedom haptic input device which uses force feedback to provide the artist with a sense of touch when sculpting. Objects are represented by sampled intensity values stored in a regularly sampled volume. As described in [12], this approach limits the amount of detail that can be achieved and requires an excessive amount of memory (minimum system requirements include 512 MB of RAM). Intensity values are low-pass filtered to reduce aliasing artifacts in the sculpted models, resulting in smoothed edges and rounded corners typical in volumetric sculpting systems. To take advantage of hardware rendering, objects are converted to polygon models using Marching Cubes [19]. However, at large volume sizes, Marching Cubes produces a prohibitive number of triangles, causing memory overload and bottlenecks in the graphics rendering pipeline which limit interactivity.

There have been a number of research papers published touting digital sculpting systems. Volume-based systems include [13], [25], [1], and [2]. Sculpting of parametric models are described in [24], [11], and [17]. However, each of these suffers from some of the limitations described above and none report sculpted results of the quality required by ILM.

Finally, it is hard to beat clay as the ultimate sculpting medium. A standard approach to creating digital character models involves sculpting a clay maquette and digitizing or scanning it to generate a digital model. There are a plethora of systems for scanning objects and generating surface models from the scanned data. Unfortunately, all scanners and digitizers have limited resolution, are unable to digitize occluded or hard-to-reach surfaces, and are subject to noise. Thus, some of the advantages of clay are lost in the input process. However, scanned models can provide good first order geometry that can be enhanced by a computer-based modeler and hence it is important that modeling systems (including Kizamu) take scanned data as input.

3. THE SCULPTING SYSTEM

Figure 2 shows a system diagram for Kizamu; individual components are discussed in detail throughout the rest of this paper. In a typical scenario the user might start by generating a basic form using CSG operations on standard shapes, proceed by sculpting the detailed object using several selectable tools, and then output the sculpted artifact as a triangle model. During sculpting, the renderer updates the image, automatically switching between several rendering methods depending on user actions and system capabilities. During idle-time, Kizamu performs a number of operations including increasing the rendering resolution of the adaptive ray caster, increasing the resolution and extent of edits from scripted tool paths, and correcting distance values for points in the ADF that are remote from the surface.

4. INTERACTION PARADIGMS

Kizamu supports a standard set of interaction tools for file manipulation, operation undos, selection, shape smoothing, etc., as well as tools for navigation and lighting control that are beyond the scope of this paper. Here we discuss several of the unique methods in Kizamu for interacting with ***volumetric*** data that exploit the properties of ADFs.

4.1 Surface Following

Given an ADF, we can reconstruct the distance value and direction to the nearest surface for any point in space. Kizamu optionally uses this information to guide the tool position and orientation, forcing the tool tip to follow the surface and to orient itself per-

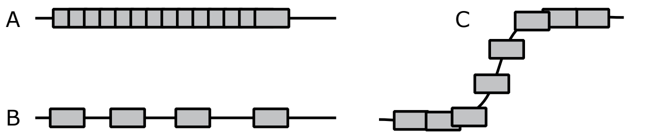


Figure 3. Discrete edits along a stroke can result in A) many redundant edits for slow tool strokes, B) broken paths for fast strokes, and C) direction-dependent results for an asymmetric tool shape.

pendicular to the surface. Surface following is accomplished by moving the tool tip in the direction of the gradient of the distance field by an amount proportional to its distance from the surface (or from some offset surface for constrained sculpting as described in Section 4.4). Surface following has proven to be particularly useful when editing a 3D shape with a 2D input device such as a mouse because it allows more intuitive control when sculpting surfaces that are not perpendicular to the viewing direction.

4.2 Bezier Tool Paths

Sculpting can be accomplished by performing a set of discrete edits along the tool path (e.g., at mouse event positions). However, as illustrated in Figure 3, this approach can be inefficient for slow mouse movements and can result in broken paths when the input device moves too quickly. If the tool shape is not axially symmetric, direction-dependent results occur (e.g., the carved path can be continuous when the major axis of the tool is parallel to the path direction and broken when it is perpendicular to the path direction). To address these issues, Kizamu can edit the ADF directly with the swept volume of a tool moved along a 3D Bezier curve. We use a Bezier clipping algorithm [21] to compute distances from the curve and then use the tool geometry to determine distances for the swept volume. For a spherical tool, distances in the swept volume are trivially and quickly computed by offsetting the distances to the centerline of the tool path. For rectangular and other simple tools, the distance computation is more involved but can still be more efficient than processing discrete edits. Using Bezier tool paths allows two levels of control: 1) Bezier paths can be automatically derived and carved during the user's input stroke and 2) a Bezier curve can be explicitly drawn onto the object's surface and edited using control points before being carved onto the surface at the user's command.

4.3 Scripting of the Tool Path

During editing, Kizamu can record edit paths in a script for later processing. The scripted paths provide three important features. First, some systems may require that the editing resolution and the extent of the tool influence on the distance field be limited in order to achieve interactive speeds. When this occurs, the scripted edit paths are used for lazy evaluation of the edits at higher resolutions and for increasing the extent of the tool influence further into the ADF during idle-time processing. Second, scripted paths provide a history of the sculpting session, allowing, for example, the models to be recreated across systems with different memory and processing power. Finally, scripted paths combined with stored intermediate versions of the ADF allow Kizamu to process multiple undos without excessive storage.

4.4 Distance-Based Constraints

Because ADFs represent the distance from the surface at any point in space, they provide an enhanced modeling interface. For example, by constraining tools to move along an offset surface, Kizamu can force the tool to remove or add material at a specified depth or height from the original surface. The computation to keep the tool on the offset surface is as simple as it is to keep the tool on the true surface (see Section 4.1). In addition, Kizamu uses auxiliary ADFs

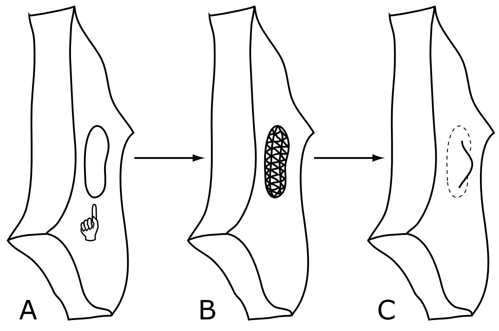


Figure 4. A region of the ADF is selected in A) and converted to triangles in B). Vertices of the triangle mesh are used for control-point editing of the shape. The ADF is then locally regenerated in C) using the edited triangle mesh.

to represent masks (e.g., cutouts) for editing and as 'force fields' that prevent the tool from entering a specified region. Finally, by using the reconstructed distance field and its gradient, ADFs trivially provide the data required for cues (e.g., force feedback for haptics) that indicate tool depth or proximity to the surface.

4.5 Control Point Editing

While clay sculpting is a powerful metaphor for digital character design, there are times when control vertex manipulation has its advantages. For example, to puff out a character's cheek, it can be easier to grab and manipulate a set of control vertices than to build out the cheek by sculpting. Thus, Kizamu provides a means for selecting a region of the ADF model, converting the local region to a triangular mesh at a user controlled LOD (see Section 7.2), manipulating the control vertices of the mesh, and locally regenerating the ADF using the edited mesh (see Figure 4).

5. GENERATION AND EDITING

5.1 Tiled Generation

Frisken *et al.* outline two methods for generating ADFs in [12]: a bottom-up algorithm that builds a fully populated octree and coalesces cells upwards when the distance field represented by a cell's 8 children is well approximated by the sampled distance values of the cell, and a top-down algorithm that recursively subdivides cells which do not pass a predicate comparing distances computed from the object's distance function with distances reconstructed from the cell's sampled distance values at 19 test points. In practice, both of these methods have limitations. The bottom-up approach requires an excessive amount of memory and too many distance computations to generate high resolution ADFs while the recursive top-down approach performs expensive octree neighbor searches and requires many redundant distance computations and reconstructions that could be shared among neighboring cells. Both approaches exhibit poor memory coherence. Times for generating an ADF with a maximum octree level of 8 (level-8 ADF) from relatively simple CSG models using the top-down generation algorithm running on a desk-top Pentium III is on the order of 20 seconds (40 seconds for a level-9 ADF of the same part). For both algorithms, generation of ADFs higher than level-9 is impractical due to long generation times and large memory requirements.

In Kizamu, we developed and implemented a generator that balances memory requirements, memory coherence, and computation. Generation times for similar CSG parts are reduced significantly to less than 1 second for a level-8 ADF and 2 seconds for a level-9 ADF (a 20 times increase in speed). Better memory utilization and reuse of distance computations allow us to generate a level-12 ADF of the same part in 7.6 seconds. It is important to note that a level-12 ADF represents a resolution range of 1 : 2^{-12} (i.e., 1 : 0.00024). Representing such detail in a regularly sampled volume would require 69 billion sample values. In contrast, a typical level-12 ADF, such as the carved slab of Figure 1, requires on the order of 2 to 3 million distance values.

In the new algorithm, the root cell is recursively subdivided to a level *L*. Cells at level *L* that require further subdivision are appended to a list of candidate cells. These candidate cells are then recursively subdivided between levels *L* and *2L*, where new candidate cells are produced and appended to the list of candidate cells, and so forth. This layered production of candidate cells proceeds until the list of candidate cells is empty.

A cell becomes a candidate for further subdivision when (1) it is a leaf cell of level *L*, or *2L*, or *3L*, etc., (2) it cannot be trivially determined to be an interior or exterior cell, (3) it does not satisfy a specified error criterion, and (4) its level is below a specified maximum ADF level.

For each candidate cell, computed and reconstructed distances are produced as needed during recursive subdivision. The distance values are stored in a ***tile***, a regularly sampled volume (i.e., a 3D array). The tile resides in cache memory and its size determines *L*. A corresponding volume of bit flags keeps track of valid distance values in the tile volume to ensure that distance values within the tile are computed and stored only once, thereby avoiding the redundant distance computations in [12]. After a candidate cell has been fully processed, valid distances in the tile volume are appended to the array of final ADF distances, and cell pointers to these distances are updated. Special care is taken at tile boundaries by initializing the tile volume and bit flag volume from neighboring cells that have already been subdivided to ensure that distances are also shared across tiles.

Tile sizes can be tuned to the CPU cache architecture. In the Pentium systems that we have tested, a tile size of 16^3 has worked most effectively. The use of the bit flag volume further enhances cache effectiveness. The final cells and distances are stored in contiguous arrays to enhance locality of reference. These arrays are allocated in large blocks as needed during generation and truncated to their required sizes when generation is complete. Because the allocation is contiguous, updating cell and distance value pointers when an ADF is moved or copied to a different base address is both fast and easy. Pseudocode for the new generation algorithm is presented below.

```
tiledGeneration(genParams, distanceFunc)
  // cells: block of storage for cells
  // dists: block of storage for final distance values
  // tileVol: temporary volume for computed and
  // reconstructed distance values
  // bitFlagVol: volume of bit flags to indicate
  // validity of distance values in tileVol
  // cell: current candidate for tiled subdivision
  // tileDepth: L (requires (2^(L+1))^3 volume – the L+1
  // level is used to compute cell errors for level L)
  // maxADFLevel: preset max level of ADF (e.g., 12)

  maxLevel = tileDepth
  cell = getNextCell(cells)
  initializeCell(cell, NULL) (i.e., root cell)

  while (cell)
    setAllBitFlagVolInvalid(bitFlagVol)
    if (cell.level == maxLevel)
      maxLevel = min(maxADFLevel, maxLevel + tileDepth)
    recurSubdivToMaxLevel(cell,maxLevel,maxADFLevel)
    addValidDistsToDistsArray(tileVol, dists)
    cell = getNextCandidateForSubdiv(cells)

initializeCell(cell, parent)
  initCellFields(cell, parent, bbox, level)
  for (error = 0, pt = cell, face, and edge centers)
    if (isBitFlagVolValidAtPt(pt))
      comp = getTileComputedDistAtPt(pt)
      recon = getTileReconstructedDistAtPt(pt)
    else
      comp = computeDistAtPt(pt)
      recon = reconstructDistAtPt(cell, pt)
      setBitFlagVolValidAtPt(pt)
    error = max(error, abs(comp - recon))
  setCellError(error)
```

```
recurSubdivToMaxLevel(cell, maxLevel, maxADFLevel)
  // Trivially exclude INTERIOR and EXTERIOR cells
  // from further subdivision
  pt = getCellCenter(cell)
  if (abs(getTileComputedDistAtPt(pt)) >
      getCellHalfDiagonal(cell))
    // cell.type is INTERIOR or EXTERIOR
    setCellTypeFromCellDistValues(cell)
    return

  // Stop subdividing when error criterion is met
  if (cell.error < maxError)
    // cell.type is INTERIOR, EXTERIOR, or BOUNDARY
    setCellTypeFromCellDistValues(cell)
    return

  // Stop subdividing when maxLevel is reached
  if (cell.level >= maxLevel)
    // cell.type is INTERIOR, EXTERIOR, or BOUNDARY
    setCellTypeFromCellDistValues(cell)
    if (cell.level < maxADFLevel)
      // Tag cell as candidate for next layer
      setCandidateForSubdiv(cell)
    return

  // Recursively subdivide all children
  for (each of the cell's 8 children)
    child = getNextCell(cells)
    initializeCell(child, cell)
    recurSubdivToMaxLevel(child, maxLevel, maxADFLevel)

  // cell.type is INTERIOR, EXTERIOR, or BOUNDARY
  setCellTypeFromChildrenCellTypes(cell)

  // Coalesce INTERIOR and EXTERIOR cells
  if (cell.type != BOUNDARY) coalesceCell(cell)
```

5.2 Bounded-Surface Generation

The generation algorithm presented in the above pseudocode does not subdivide interior or exterior cells. This surface-limited generation reduces memory requirements while assuring accurate representation of the distance field in boundary cells, which is sufficient for processing (e.g., rendering) that is limited to regions adjacent to the surface. However, many of the interaction methods discussed in Section 4 such as surface following, the use of distance-based constraints, and force feedback, require the distance field to be accurate for some distance from the surface. Under these circumstances the generation algorithm is modified so that exterior (and/or interior) cells are not coalesced unless their parent cells satisfy a maximum error constraint whenever those parent cells are within a bounded region defined by a specified minimum distance from the surface.

5.3 Editing

In Kizamu, sculpting is performed on the ADF using a local editing process that extends the algorithm of [12] to use tiled generation. The use of tiled generation decreases edit times by a factor of 20 and changes in the ADF octree cell data structure (e.g., support for 16-bit distance values) reduce memory requirements for cells and distance values by a factor of 2. These advances make it possible to perform highly detailed sculpting. Figures 1 and 5 present several models created using Kizamu. The sculpted models were carved with procedurally generated tool paths (we never said that we were sculptors) using techniques related to procedural texturing [10]. Space limitations prevent a full description of this technique but we emphasize that the detail in these images is purely geometric: achieving such surface detail using existing modeling systems would be painful if not impossible.

To accommodate available resources on different computer systems, Kizamu can limit both the sculpted resolution and the volumetric extent of the tool influence to ensure interactivity. As described in Section 4.3, Kizamu maintains a script of sculpting operations and intermediate versions of the ADF which can be used during idle-time processing to increase the sculpting resolution and extend the tool influence into the ADF. The relatively small stroke numbers reported in Figures 1 and 5 show that the memory cost of maintaining the stroke history is reasonable.

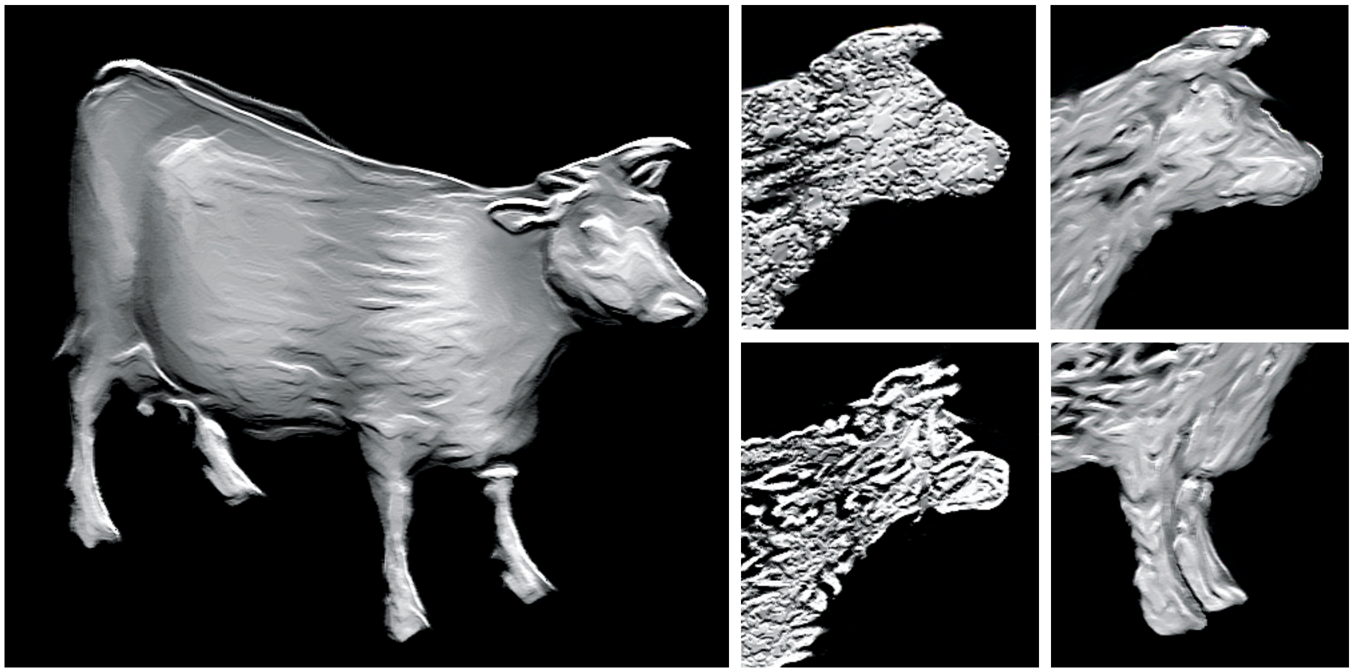
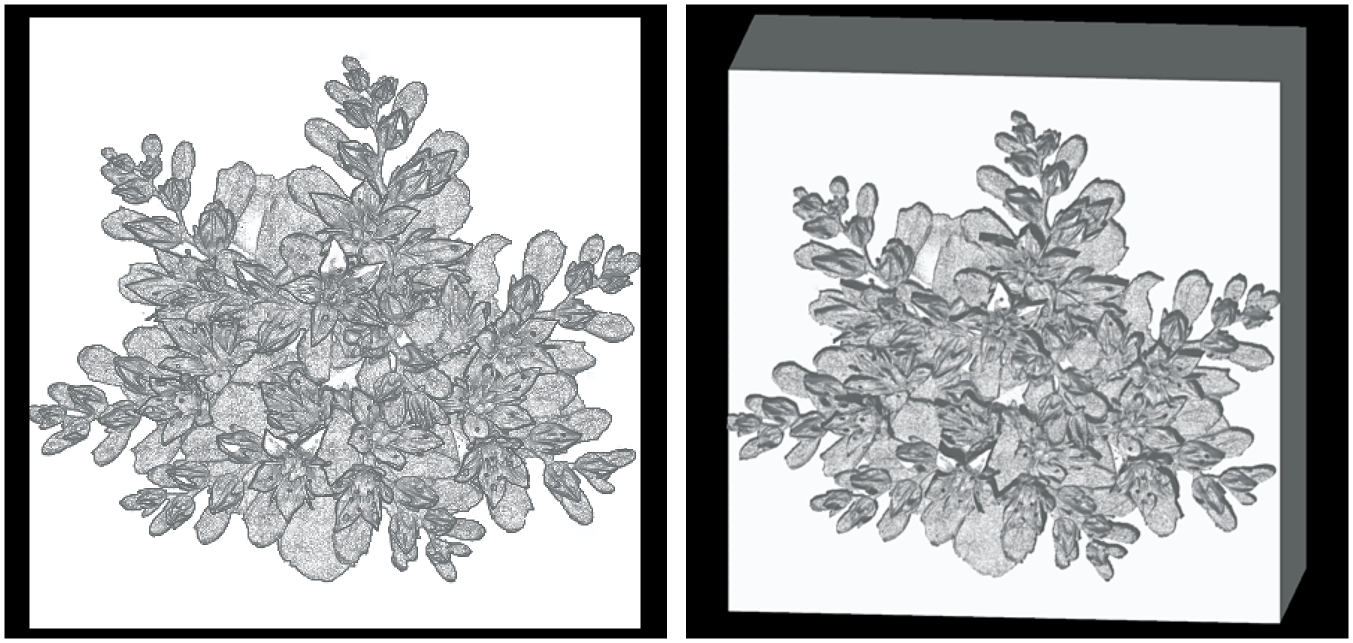


Figure 5A. A 3D cow sculpted using Kizamu with four different procedural methods for determining chisel strokes. The procedural sculpting used the distance field and cell size of an ADF generated from a low resolution (5800 triangles) model. Simple Phong illumination was used by the renderer; all texture is purely geometric. Each cow is a level-10 ADF, requiring approximately 150 MB of storage, and was carved using 21,063 chisel strokes.



Figures 5B. A bas-relief carving on a slab. The 3D geometry was generated from a black and white image of a flower (photograph courtesy of John Arnold) using the algorithm described in Section 7.1 for converting range data to an ADF. The highly detailed carving is represented as a level-10 ADF requiring 186 MB of storage.

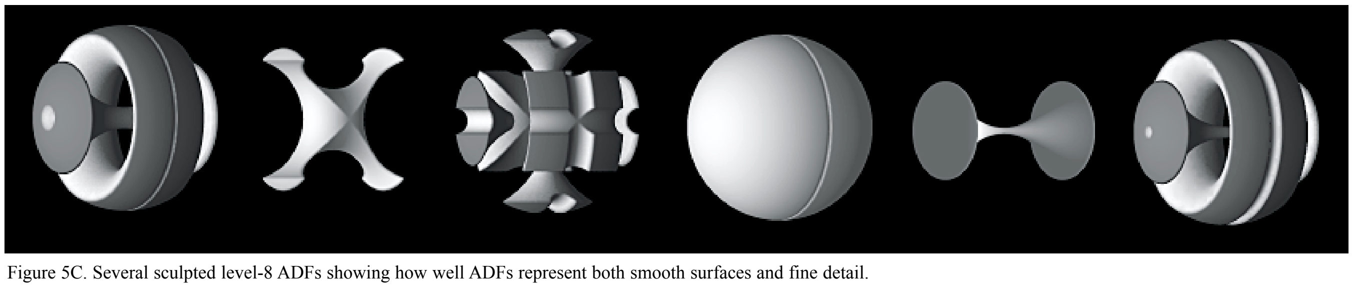


Figure 5C. Several sculpted level-8 ADFs showing how well ADFs represent both smooth surfaces and fine detail.

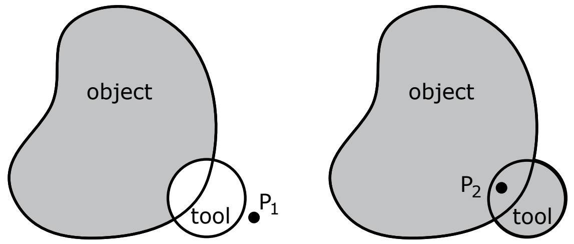


Figure 6. Distances computed using min/max CSG operators are inaccurate at P_1 for the differencing operator and at P_2 for the additive operator.

5.4 Correcting the Remote Field

ADFs are sculpted by applying CSG operations to the object and tool distance fields. For example, following the positive-inside / negative-outside sign convention in [12], the sculpted distance at a point *p* for a differencing tool would be $dist(p) = min(dist_{obj}(p), -dist_{tool}(p))$ and the sculpted distance at *p* for an additive tool would be $dist(p) = max(dist_{obj}(p), dist_{tool}(p))$. As pointed out by Breen *et al*. in [5], and illustrated in Figure 6, the use of such min/max operators can result in inaccurate distance values in the distance field at points remote from the sculpted surface. For example, the point P_1, when using a differencing tool, and the point P_2, when using an additive tool, would both be assigned distances to surfaces that do not exist in the sculpted object, resulting in an inaccurate field at these points. When processing only requires a correct representation of the distance field at the surface (e.g., rendering), these inaccurate values have no effect. However, some of the interaction techniques used in Kizamu require an accurate representation of the distance field away from the surface. Hence, Kizamu corrects the remote distance field during system idle-time and on-demand near the tool during editing.

There are a number of approaches available for correcting the remote distance field given accurate distances close to an object's surface. Breen *et al.* [5] use a fast marching method derived from level set techniques to propagate distances outward from a narrow band of correct values near the surface. Rockwood has proposed an approach that holds distance values at the zero-valued iso-surface and the distance field boundaries constant and uses simulated annealing to smooth out (but not correct) the field in between. Neither Breen nor Rockwood apply their methods to adaptive grids and both methods are relatively slow. Kizamu employs a correction method that extends the region growing Euclidean distance transforms prevalent in the field of computer vision (see [7] for a good review of such approaches) to the adaptively sampled grid.

6. FAST RENDERING

6.1 Fast Global Rendering

We have developed and implemented two efficient new algorithms for converting ADFs to triangle and point models that allow Kizamu to take advantage of standard hardware during global view changes such as rotation. Given current standard hardware such as the NVIDIA GeForce2, these new algorithms provide truly interactive manipulation of ADFs. First, ADFs can be converted to triangle models using the method described in Section 7.2 and rendered through OpenGL. Conversion times are quite fast: Kizamu can generate a triangle model with 200,000 triangles from a level-9 ADF in 0.37s on a Pentium IV processor. Models at lower LODs can be generated even faster (less than 10ms for a 2000-triangle model). These fast conversion times enable models to be created on-the-fly with imperceptible delays when navigation tools are selected.

Alternatively, Kizamu can use point-based rendering for global view changes, taking advantage of the current trend in computer graphics towards hardware-assisted point rendering [22].

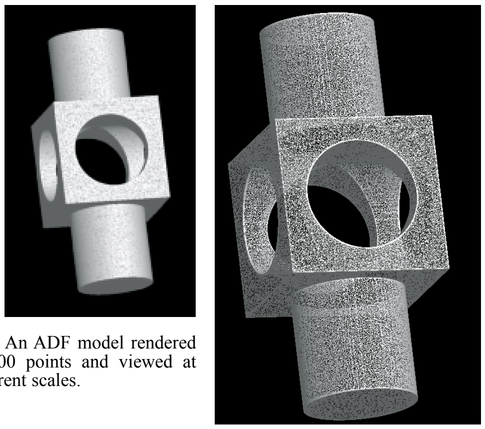


Figure 7. An ADF model rendered as 800,000 points and viewed at two different scales.

Kizamu currently renders points using the OpenGL pipeline. The point generator can produce 800,000 Phong illuminated points in 0.2 seconds (and less than 0.12 seconds for points lit using hardware). These points can be rendered interactively on desktop systems and, like triangle models, can be created on demand when navigation tools are selected. The basic algorithm for generating points from an ADF is outlined in the pseudocode below. Figure 7 shows an ADF model rendered as 800,000 points. The availability of the distance field and the octree data structure give ADFs unique advantages for generating points. For example, a point randomly generated within a cell can be trivially projected onto the surface in one step using the distance value and gradient at that point. Furthermore, a cell's size can be used to determine both the number and size of points to be generated in that cell, thereby permitting hierarchical rendering methods [22].

```
// Algorithm:
// 1. Determine the number of points to generate in
//    each boundary leaf cell
//    a. Compute an estimate of the object's surface
//       area within each boundary leaf cell areaCell
//       and the total estimated surface area of the
//       object, areaObject = Σ areaCell
//    b. Set the number of points in each cell nPtsCell
//       proportional to areaCell / areaObject
// 2. For each boundary leaf cell in the ADF
//    a. Generate nPtsCell random points in the cell
//    b. Move each point to the object's surface using
//       the distance and gradient at the point

generatePoints(adf, points, nPts, maxPtsToGen)
  // Estimate object's surface area within each
  // boundary leaf cell and the total object's
  // surface area
  for (areaObject = 0, level = 0 to maxADFLevel)
    nCellsAtLevel =
      getNumBoundaryLeafCellsAtLevel(adf, level)
    areaCell[level] = sqr(cellSize(level))
    areaObject += nCellsAtLevel * areaCell[level]

  // nPtsCell is proportional to areaCell / areaObject
  for (level = 0 to maxADFLevel)
    nPtsAtLevel[level] = maxPtsToGen *
      areaCell[level] / areaObject

  // For each boundary leaf cell, generate cell points
  // and move each point to the surface
  for (nPts = 0, cell = each boundary leaf cell of adf)
    nPtsCell = nPtsAtLevel[cell.level]
    while (nPtsCell--)
      pt = generateRandomPositionInCell(cell)
      d = reconstructDistAtPt(cell, pt)
      n = reconstructNormalizedGradtAtPt(cell, pt)
      pt += d * n
      n = reconstructNormalizedGradtAtPt(cell, pt)
      setPointAttributes(pt, n, points, nPts++)
```

Because point generation is very fast, we are exploring real-time ***view-dependent*** point generation for every view change (e.g., while rotating) that culls both back-facing cells and points, and uses cell gradients to generate more points on silhouette edges.

6.2 Fast Local Rendering

Kizamu uses two methods to locally update the image in the sculpted region depending on system resources: adaptive and non-adaptive ray casting for software-based rendering, and local triangle regeneration for hardware-based rendering. Point models do not represent fine sculpted detail as well as triangle models or images derived from ray casting and therefore are not used for local rendering.

When there is sufficient memory to maintain both the ADF and its corresponding triangle model, Kizamu can locally update the triangle model during sculpting. Each boundary cell references (via an index) the triangles that represent the surface in the cell to enable local updating. When a cell is affected by sculpting, its associated triangles are deleted and new ones regenerated on-the-fly. This approach is very fast but requires memory to maintain the dynamic triangle model as well as additional memory per cell for triangle indexing.

Kizamu provides ray casting for high quality rendering of the sculpted surface. During sculpting, the ray cast image is updated locally within the region affected by the tool. For systems that cannot perform software-based ray casting fast enough for interactive updating, Kizamu uses an adaptive ray casting approach. In the spirit of [15], the image region to be updated is divided into a hierarchy of image tiles and subdivision of the image tiles is guided by a perceptually based predicate. Pixels within image tiles of size greater than 1x1 are bilinearly interpolated to produce the image. For each image tile, rays are cast into the ADF at tile corners and intersected with the surface using the linear intersection method outlined in [12]. The predicate used to test for further subdivision follows Mitchell [20] and more recently [4], individually weighting the contrast in red, green, and blue channels and the variance in depth-from-camera across the image tile. The adaptive approach typically achieves a 6:1 reduction in rendering times over firing 1 ray per pixel and more than a 10:1 reduction when the image is supersampled for antialiasing. Figure 8 shows an image rendered using adaptive ray casting and the rays that were cast to produce the image.

7. FITTING KIZAMU INTO THE CHARACTER ANIMATION PIPELINE

Any sculpting system using a novel data representation for digital character design is useless to a production house like ILM unless it fits into their extensive existing character animation pipeline. Hence, Kizamu inputs models from several types of representations (including triangle models, implicit functions, CSG models, Bezier patches, and scanned data) and outputs triangle models. Converting most of the above representations to ADFs is treated in [12]. Here, we focus on two important new developments: 1) a method for generating ADFs from scanned range data which advances the state of the art and 2) a new method for triangulating ADFs that generates topologically consistent LOD triangle models in a fraction of a second.

7.1 Input from Range data

There are several commercial systems available for converting scanned range data to triangle models. One approach for importing this data into Kizamu would be to use such a system to convert the scanned data to a triangle model and then convert the triangle model to an ADF. However, our experience has been that the triangle models produced by these systems are often problematic, containing holes, flipped triangles, and overlapping surfaces. Instead, Kizamu extends recent research to generate ADFs directly from scanned data.

Several recent research papers have presented methods for converting scanned data to triangle models that make use of signed distance fields for more robust, water-tight surface reconstructions. The method of Curless and Levoy [8] is most frequently cited and is the basis for the data conversion method used by the Cyberware scanning system. Wheeler [27] recently extended [8] to use a true Euclidean distance function and a more volume-centric approach. A full description and comparison of these methods is beyond the scope of this paper. However, we outline Wheeler's approach since it is most closely related to the algorithm that was developed and implemented in Kizamu.

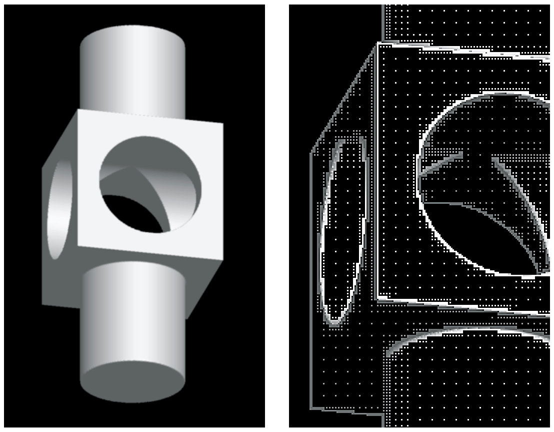


Figure 8. An ADF rendered using adaptive ray casting. The right image shows a magnified view of the rays cast to render the left image.

In Wheeler's algorithm, scanned data consists of a set of overlapping 2D scanned range images taken from different viewpoints. Each range image is converted to a range surface triangle mesh and all of the triangle meshes are imported into the same coordinate system. Wheeler then creates an octree of the model's distance field in a top-down manner by computing signed distances from each sample point to each range surface and combining these distances using a probability-weighted function. The probability function depends on the angle between each contributing triangle and the original scan direction, the location of the triangle relative to the edge of its source range image, the degree of agreement between distances computed for overlapping range surfaces, and possibly other factors. All cells containing the object's boundary are subdivided to the same octree level (similar to the 3-color octree described in [12]). As a last step, Marching Cubes is run over all the boundary leaf cells to generate a closed (assuming it makes use of volume boundaries), water-tight triangle surface from the signed distance volume.

Kizamu adopts Wheeler's approach but enhances the algorithm using ADFs, replacing his top-down generation of a 3-color octree with tiled generation of an ADF. In ADFs, the surface within each boundary cell is represented as a trilinear field; consequently, boundary cells do not require subdivision in flat or near-flat regions of the surface, resulting in a significant savings over the 3-color octree in both memory requirements and the number of distance computations [12]. Once scanned data is imported into Kizamu, the sculpting system can be used to correct problems in the model due to occluded regions, detail lost because of resolution limits or scanner noise, and rough or broken surfaces at seams between range images. Finally, if desired, the ADF can be converted into a triangle model from the adaptive grid using the approach described in Section 7.2.

7.2 Conversion to Triangles

Kizamu employs a new method to triangulate ADFs for rendering, control point-based editing, and conversion to output models that can be integrated into an existing animation pipeline. The method produces topologically consistent (i.e., orientable and closed) triangle models from an implicit function sampled on an adaptive grid such as the ADF octree. It is very fast, producing triangle models in a fraction of a second and can be used in conjunction with the ADF data structure to produce LOD models as described below.

The new method is based on the SurfaceNets algorithm that was developed as an alternative to Marching Cubes for building globally smooth but locally accurate triangle models from binary volume data [14]. Unlike [14], the enhanced SurfaceNets algorithm in Kizamu generates a triangle mesh from distance values sampled on an ***adaptive*** grid. It produces 'high quality' triangles (i.e., triangles that are close to equilateral) whose vertices lie on the object's surface. Further, as discussed below, the method easily deals with the crack problems typically associated with adaptive grids. The basic 3-step method is as follows:

1. Each boundary leaf cell of the ADF is assigned a vertex that is initially placed at the cell center.

2. Vertices of neighboring cells are joined to form triangles using the following observations: 1) a triangle joins the vertices of 3 neighboring cells that share a common edge (hence triangles are associated with cell edges); 2) a triangle is associated with an edge only if that edge has a zero crossing of the distance field; 3) the orientation of the triangle can be derived from the orientation of the edge it crosses, and 4) because cells have 12 edges, any cell can have at most 12 triangles associated with it. In order to avoid making redundant triangles, we consider 6 of the 12 possible edges for each cell (i.e., the cell's up-right, down-left, up-front, down-back, front-right, and back-left edges). (Note for the purist: if the cell contains surface saddle points, additional interior triangles may be required to correctly represent the surface topology within the cell [6]. However, treatment of this issue is beyond the scope of this paper.)

3. After all the triangles have been formed and we have a topologically consistent mesh, triangle vertices are moved towards the surface (in the direction of the ADF gradient with a step size equal to the distance value) to improve the mesh accuracy. Each vertex is then moved over the surface towards the average position of its neighbors to improve triangle quality.

Pseudocode for the enhanced SurfaceNets algorithm follows:

```
triangulateADF(adf)
  // vertices: storage for vertices
  // triangles: storage for triangles

  // Initialize triangles vertices at cell centers
  // and associate each vertex with its cell

  for (cell = each boundary leaf cell of adf)
    v = getNextVertex(vertices)
    associateVertexWithCell(cell, v)
    v.position = getCellCenter(cell)

  // Make triangles. Each cell edge joins two cell
  // faces face1 and face2 which are ordered to ensure
  // a consistent triangle orientation (see EdgeFace
  // table below). For a given cell edge and face,
  // getFaceNeighborVertex returns either the vertex of
  // the cell's face-adjacent neighbor if the
  // face-adjacent neighbor is the same size or larger
  // than the cell, OR, the vertex of the unique child
  // cell (uniqueness is guaranteed by a
  // pre-conditioning step – see text below) of the
  // face-adjacent neighbor that is both adjacent to
  // the face and has a zero-crossing on the edge

  for (cell = each boundary leaf cell of adf)
    for (edge = cell's up-right, down-left, up-front,
    down-back, front-right, and back-left edges)
      if (surfaceCrossesEdge(edge))
        face1 = EdgeFace[edge].face1
        face2 = EdgeFace[edge].face2
        v0 = getCellsAssociatedVertex(cell)
        v1 = getFaceNeighborVertex(face1, edge)
        v2 = getFaceNeighborVertex(face2, edge)
        t = getNextTriangle(triangles)
        if (edgeOrientation(edge) > 0)
            t.v0 = v0, t.v1 = v1, t.v2 = v2
        else
            t.v0 = v0, t.v1 = v2, t.v2 = v1

  // Relax each vertex to the surface and then along
  // the tangent plane at the relaxed position towards
  // the average neighbor position

  for (each vertex)
    v = getVertexPosition(vertex)
    u = getAveragePositionOfNeighborVertices(vertex)
    cell = getVertexCell(vertex)
    d = reconstructDistAtPt(cell, v)
    n = reconstructNormalizedGradtAtPt(cell, v)
    v += d * n
    v += (u - v) - n · (u - v)

// -------------------------------
// EdgeFace table:
//    edge           face1    face2
//    ----------     -----    -----
//    up-right       up       right
//    down-left      down     left
//    up-front       up       front
//    down-back      down     back
//    front-right    front    right
//    back-left      back     left
```

Most algorithms for triangulating sampled implicit functions generate triangle vertices ***on cell edges and faces*** [19][3][16]. As illustrated in 2D in Figure 9A, this approach can cause cracks in the triangulated surface at the boundaries between cells of different sizes in an adaptive grid because the interpolated vertex position of one cell may not match the interpolated vertex position of its connected neighboring cell. This problem has been addressed in several ways [26][23] but in general adds significant complexity to the triangulation algorithm.

In contrast, in the enhanced SurfaceNets algorithm, triangle vertices are generated ***at cell centers***: triangles cross the faces and edges of different sized cells. Thus, the type of cracks illustrated in Figure 9A do not appear. However, like other triangulation methods, our algorithm produces cracks when the number of edge crossings between two neighboring sets of cells differs as illustrated in Figure 9B. The 2D cell on the top in Figure 9B has no zero-crossings on its bottom edge while the edge-adjacent neighbors have a total of two zero crossings for the same edge. This type of crack can be prevented by a simple pre-conditioning of the ADF during generation or prior to triangulation. In 3D, the pre-conditioning step compares the number of zero-crossings of the iso-surface for each face of each boundary leaf cell to the total number of zero-crossings for faces of the cell's face-adjacent neighbors that are shared with the cell. When the number of zero-crossings are not equal for any face, the cell is subdivided using distance values from its face-adjacent neighbors until the number of zero-crossings match. During pre-conditioning, the cell on the top in Figure 9B would be further subdivided. Because we have observed that

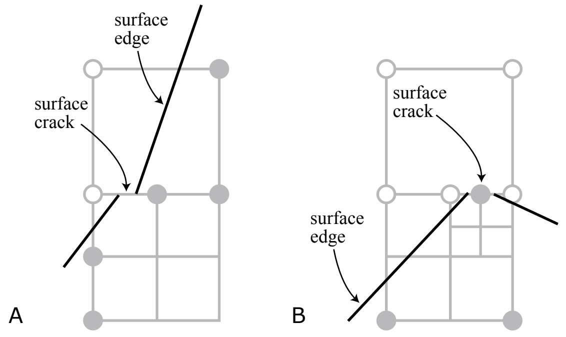


Figure 9. Adaptive grids can result in A) cracks in the surface edge which ***do not occur*** with the enhanced SurfaceNets algorithm and B) cracks in the surface edge which occur ***very rarely*** and can be prevented by a simple pre-conditioning step.

the type of crack illustrated in Figure 9B occurs rarely, Kizamu provides an optional cell pre-conditioning step when converting an ADF to triangles.

Kizamu also exploits the hierarchical data structure of ADFs to produce LOD models with the enhanced SurfaceNets algorithm. Rather than seeding triangle vertices in boundary leaf cells, the hierarchical data structure is traversed and vertices are seeded into boundary cells whose maximum error (computed during generation or editing) satisfies a user-specified threshold and cells below these cells in the hierarchy are ignored. The error threshold can be varied continuously enabling fine control over the number of triangles generated in the LOD model. Unlike most triangle decimation algorithms, the time to produce an LOD model is proportional to the number of vertices in the output mesh rather than the size of the input mesh. Generation times are at least an order of magnitude faster than the state-of-the-art: we generate a 200,000 triangle model in 0.37 seconds and a 2000 triangle model in less than 0.010 seconds on a desktop system with a Pentium IV processor.

8. CONCLUSIONS AND FUTURE WORK

The Kizamu system is a first step towards integrating ADFs, a powerful new shape representation, into the production pipeline for digital character design and animation. Kizamu is an interactive system for sculpting high resolution digital characters that is intuitive to use and efficient in memory usage and processing. We have presented several new algorithms and approaches that were required to take ADFs beyond the concept stage introduced in [12] and into a practical, working system. These include: innovations in the volumetric sculpting interface that take advantage of the distance field and provide more control to the artist; efficient generation and editing algorithms with reduced memory requirements, better memory coherency, and reduced computation; several new rendering approaches that take advantage of hardware acceleration in standard PCs; and a new and very fast method for generating topologically consistent LOD triangle models from ADFs.

We are currently exploring a number of ways to enhance Kizamu as a sculpting tool as well as to extend the use of ADFs in digital animation beyond sculpting. We have presented a method for generating triangle models from ADFs so that models created with Kizamu can be used in the character animation pipeline. To increase compatibility, we are looking into methods for generating NURBS as well. In particular, because animators require well-placed control vertices for better control, we are exploring user-guided placement of control curves onto the ADF surface to guide the NURBS generation. In addition, we are investigating a number of methods for deforming and animating ADFs, including free form deformations, finite element methods, and skeleton-based methods. Since ADFs are a volumetric representation they represent object interiors, which is often required to produce physically plausible deformation. Finally, Kizamu currently has limited functionality to take advantage of ADF's ability to represent volumetric characters (see Figure 10). Since we believe that the ability to sculpt volumes has enticing possibilities, we plan to extend Kizamu to provide improved volume rendering and a means for integrating these volumes into the animation pipeline.

9. ACKNOWLEDGEMENTS

We are indebted to Joe Letteri, Visual Effects Supervisor at ILM, for his interest in this project and many insights that helped set priorities. Thanks to John Arnold who provided the photograph used to generate the ADF in Figure 5B, to Ray Jones for helpful comments on this paper and insights into Bezier Clipping, and to Jackson Pope who coded an initial implementation of the triangulation algorithm.

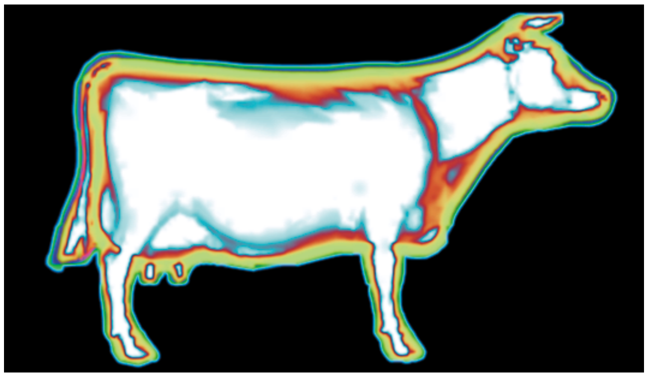


Figure 10. An ADF character volumetrically rendered.

10. REFERENCES

[1] R. Avila and L. Sobierajski, "A haptic interaction method for volume visualization", Proc. IEEE Visualization'96, pp. 197-204, 1996.
[2] J. Baerentzen, "Octree-based volume sculpting", Proc. Late Breaking Hot Topics, IEEE Visualization'98, pp. 9-12, 1998.
[3] J. Bloomenthal, "Polygonization of Implicit Surfaces", Computer Aided Geometric Design, 5(00): pp.341-355, 1988.
[4] M. R. Bolin and G. W. Meyer, "A perceptually based adaptive sampling algorithm", Proc. SIGGRAPH'98, pp. 299--310, 1998.
[5] D. Breen, S. Mauch and R. Whitaker, "3D scan conversion of CSG models into distance volumes", Proc. 1998 IEEE Symp. on Volume Visualization, pp. 7-14, 1998.
[6] P. Cignoni, F. Ganovelli, C. Montani, and R. Scopigno, "Reconstruction of Topologically Correct and Adaptive Trilinear Isosurfaces", Computers & Graphics, Elsevier Science, Vol. 24, no. 3, pp. 399-418, 2000.
[7] O. Cuisenaire, "Distance Transformations: Fast Algorithms and Applications to Medical Image Processing", Ph.D. thesis, Universite Catholique de Louvain, 1999.
[8] B. Curless and M. Levoy., "A Volumetric Method for Building Complex Models from Range Images", Proc. SIGGRAPH `96, pp. 303-312, 1996.
[9] T. DeRose, M. Kass, T. Truong, "Subdivision surfaces in character animation", Proc. SIGGRAPH '98, pp. 85-94, 1998.
[10] D. Ebert, F. Musgrave, D. Peachy, K. Perlin, and S. Worley, *Texturing and Modeling a Procedural Approach*, Academic Press, 1998.
[11] B. Fowler, "Geometric manipulation of tensor product surfaces", Proc. of the 1992 Symposium on Interactive 3D Graphics, pp. 101-108, 1992.
[12] S. Frisken, R. Perry, A. Rockwood, and T. Jones, "Adaptively Sampled Distance Fields: A General Representation of Shape for Computer Graphics", Proc. SIGGRAPH '00, pp. 249-254, 2000.
[13] T. Galyean and J. Hughes, "Sculpting: An Interactive Volumetric Modeling Technique", Proc. SIGGRAPH '91, pp. 267-274, 1991.
[14] S. Gibson, "Using distance maps for smooth surface representation in sampled volumes", Proc. 1998 IEEE Volume Visualization Symposium, pp. 23-30, 1998.
[15] B. Guo, "Progressive Radiance Evaluation Using Directional Coherence Maps", Proc. SIGGRAPH '98, pp. 255-266, 1998.
[16] D. Karron and J. Cox, "New findings from the SpiderWeb algorithm: toward a digital morse theory", Proc. Vis. in Biomedical Computing , pp.643-657, 1994.
[17] A. Khodakovsky and P. Schroder, "Fine Level Feature Editing for Subdivision Surfaces", in ACM Solid Modeling Symposium, 1999.
[18] J. Letteri, Visual Effects Supervisor, Industrial Light and Magic, personal communication.
[19] W. Lorensen and H. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm", Proc. SIGGRAPH '87, pp.163-169, 1987.
[20] D. Mitchell, "Generating antialiased images at low sampling densities", Proc. SIGGRAPH '87, pp. 65-72, 1987.
[21] T. Nishita, T.W. Sederberg and M. Kakimoto, "Ray tracing trimmed rational surface patches", Proc. SIGGRAPH '90, pp. 337-345, 1990.
[22] S. Rusinkiewicz and M. Levoy, "Qsplat: A Multiresolution Point Rendering System for Large Meshes", Proc. SIGGRAPH'00, pp. 343-352, 2000.
[23] R. Shekhar, E. Fayyad, R. Yagel, and J. Cornhill, "Octree-based decimation of Marching Cubes Surfaces", in Visualization'96, pp. 335-342, 1996.
[24] D. Terzopoulos, and H. Qin, "Dynamic NURBS with geometric constraints for interactive sculpting", ACM Transactions On Graphics, 13(2), pp. 103-136, 1994.
[25] S. Wang and A. Kaufman, " Volume sculpting", 1995 Symposium on Interactive 3D Graphics, ACM SIGGRAPH, pp. 151-156, 1995.
[26] R. Westermann, L. Kobbelt, and T. Ertl, "Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces", The Visual Computer, 15, pp. 100-111, 1999.
[27] M. Wheeler, "Automatic Modeling and Localization for Object Recognition", Ph.D. thesis, Carnegie Mellon University, 1996.
[28] B. Wyvill, C. McPheeters, and G. Wyvill, "Animating soft objects" The Visual Computer, 2(4):235--242, 1986.