COLLAGE SCULPTURES

A Thesis

by

ELIZABETH GRACE NEMMERT MUHM

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2011

Major Subject: Visualization

COLLAGE SCULPTURES

A Thesis

by

ELIZABETH GRACE NEMMERT MUHM

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2011

Major Subject: Visualization

ABSTRACT

Collage Sculptures. (December 2011)

Elizabeth Grace Nemmert Muhm, B.S., University of Washington

Chair of Advisory Committee: Dr. Ergun Akleman

In this thesis, I develop a program to automatically assemble *collage sculptures*, sets of arbitrary, non-overlapping elements arranged to fill out a recognizable target shape according to a set of procedural rules. A user provides the target and element shapes and the program procedurally places the elements in spherical holes in the target space. A signed distance function defined over the target space keeps track of the remaining holes to fill. Elements are preprocessed to determine the size of their smallest enclosing bounding sphere. They are placed in holes based on the radius of their bounding sphere. After each placement, the signed distance function is efficiently updated to account for the newly added element. Elements are placed from largest to smallest, filling the space to a predefined threshold. To demonstrate this program, I generated a number of collage sculptures. In accordance with our procedural rules, the elements in the resulting collage sculptures recognizably represent the target shape, do not overlap, are not deformed from their original shape, and display variety in size, position, and orientation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

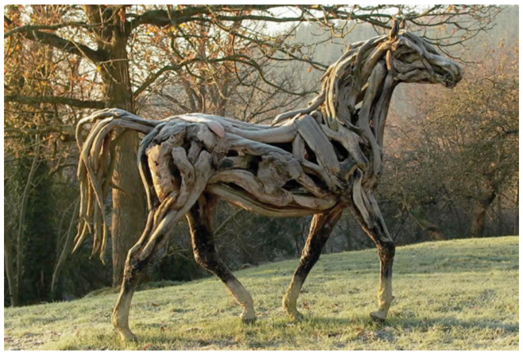
The goal of this work is to create *collage sculptures*, 3D computer graphics sculptures composed of arbitrary objects pieced together in a non-overlapping arrangement according to a set of procedural rules. We present a new method for automatically assembling user-given elements into a user-given shape to create such a collage sculpture. The process relies on a voxel-based discretization of the target space to keep track of open positions in which to place elements. Elements are added to the sculpture one by one, and with each addition, the data structure describing the space is efficiently updated. The resulting collage sculptures recognizably represent the target shape, and this is achieved without deforming the elements or allowing overlap of elements.

I.1. Motivation

In the world of fine art, we see many examples of collage structures, one shape comprised of unique elements, such as the paintings of Arcimboldo and the sculptures of Heather Jansch [2], [19]. See Fig. 1. Assemblage art in particular combines found objects, elements not originally intended as art materials, into artistic compositions. Work of Jean Dubuffet and Pablo Picasso exemplifies this style [7], [27], [29]. Other collage examples include the work of illustrator Hanoch Piven who creates caricatures using found objects to suggest facial features, as well as the work of Ergun Akleman who assembles found internet images to create caricatures [1], [28]. See Fig. 2. Ad-

The journal model is *IEEE Transactions on Visualization and Computer Graphics.*

(a) Vertumnus, 1590


(b) Apollo, 2005

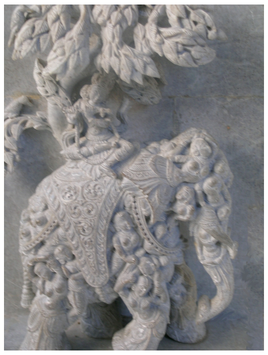
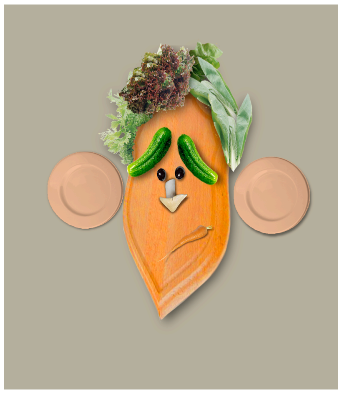
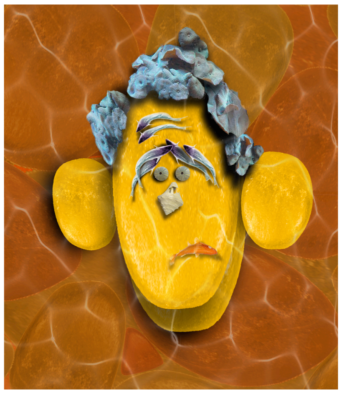

(c) Unknown

Fig. 1. Examples of collage art in fine art: (a) A collage image by Giuseppe Arcimboldo [2]. (b) A collage sculpture by Heather Jansch [19]. (c) A collage sculpture from India.


(a) George Bush, 2011


(b) George Bush, 2011

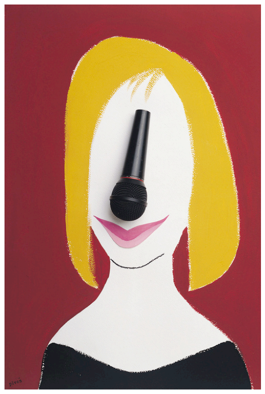
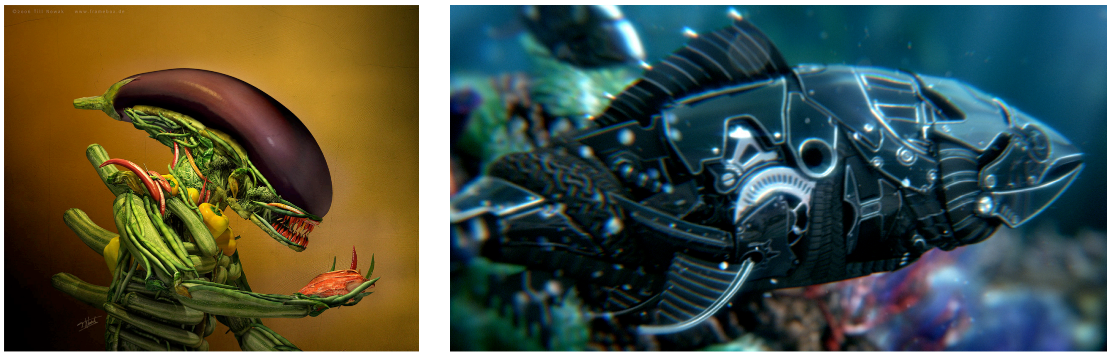

(c) Barbra Streisand, 1994

Fig. 2. Caricature collage art: (a), (b) Caricature collages made by Ergun Akleman [1]. (c) Caricature collage by Hanoch Piven [28].

ditionally, George Hart creates geometric sculptures out of interesting elements such as plastic forks and spoons [14]. In all these varied examples, the composite work takes on a new meaning derived from the form and juxtaposition of its elements.

Following suit, in the realm of computer graphics, artists have created 3D collage objects, often painstakingly modeling them by hand. Fitting the elements together is a difficult challenge. For, it is time consuming to plan the best location for each element and tedious to position them without overlap. Fig. 3 provides examples of 3D collage sculptures from computer graphics [25], [5]. To create both examples, artists modeled the characters by hand, planning each with detailed concept art.

More recently, researchers have developed algorithms to automatically place elements in target shapes [3], [10], [12], [16], [18], [22], [24], [26]. Here, we present an alternate algorithm that efficiently creates non-overlapping arrangements and offers a new aesthetic for the collage sculptures created.


(a) Salad, 2006

(b) A character from ShapeShifter, 2010

Fig. 3. Examples of collage art in computer graphics, modeled by hand: (a) A collage sculpture by Till Nowak [25]. (b) A collage character by Charlex Studio [5].

I.2. Introduction

Our algorithm for automatically assembling elements into a target shape is driven by the following aesthetic rules. (1) *Visually clarify target shape.* The arrangement captures enough detail of the target shape to be recognizable. (2) *Avoid element overlap.* The arranged elements will not overlap at any point. (3) *Avoid element deformation.* In the tradition of assemblage art, the algorithm uses the elements as they are found. It will not deform the objects. (4) *Display variety.* The arranged elements will display variety in element size, orientation, and position.

To create a collage sculpture, users must only specify the target shape and a set of element shapes. The program then automatically positions and orients those element shapes to best fill out the target shape. The program keeps track of empty space left to be filled and adds elements one at a time based on the amount of space available. The space filled by the target shape is discretized into a voxel structure over which a signed distance function is defined. Since each signed distance function value represents the distance to the closest surface point from that voxel, a sphere with a radius equal to the signed distance function value could be centered at that voxel and just touch the closest surface point.

Using the idea of packing tangent spheres to avoid overlap, the elements are preprocessed to find their smallest containing bounding sphere. One at a time, elements are added into the collage sculpture and centered at a voxel whose current signed distance function value is greater than or equal to the sphere's radius. After each element is added, the signed distance function values are efficiently updated to account for the new element, and the process repeats. Originally, the signed distance function measures the distance from each voxel to the surface of the target shape. As each element is added, the surface definition is expanded to include the surface of the newly added element. Thus each voxel stores the nearest distance either to the original target surface or the nearest element depending on whichever is closer.

The rest of this thesis is organized as follows: Chapter II summarizes related work. Chapter III explains the process for creating collage sculptures. Chapter IV details the element placement algorithm. Chapter V describes the implementation and results, and Chapter VI concludes and discusses future work.

CHAPTER II

RELATED WORK

With his composite head paintings, Guiseppe Arcimboldo (1527-1593) introduced the idea of using collages of elements to represent a greater form. “No other artist before him had ever thoroughly explored the possibilites of the subject and composition techniques that were to create his so-called ‘composed heads’” [8]. Collage resurfaced in the assemblage art of the 20th century. Artists such as Picasso, Dubuffet, and Duchamp incorporated found objects into their compositions [29]. Now researchers have taken this inspiration to the realm of computer science and used it to generate a new body of work, taking advantage of computers to process the difficult task of assembling elements to fit a space. This chapter summarizes work related to our Collage Sculptures process and helps explain the progression of ideas that lead to this thesis. It is divided into four sections, each of which covers a different category of influence on the work. Section II.1 highlights the 2D image placing methods that preceded the 3D element placing methods described in Section II.2. Section II.3 describes the sphere packing problem, and II.4 summarizes shape description methods.

II.1. 2D Image Placing Methods

Computer algorithms have been explored as a means to creating collage art. Their processing power can be leveraged to solve the difficult challenge of placing elements while maintaining the essence of the larger form. A number of computer graphics methods have been developed for creating 2D mosaics from tile elements. Jigsaw Image Mosaics takes a database of input images of arbitrary shapes and sizes to use as tiles to fill an arbitrary target shape [21]. The problem statement of Collage

Sculptures is nearly the 3D analog of that of Jigsaw Image Mosaics. However, their solution allows small element deformations and overlap which violates two of our procedural placing rules. Their algorithm relies on shape descriptions of the tiles and target that are achieved with active contours. It then uses a best first search for fitting tiles in open spaces that minimize an energy function accounting for color, overlap, gaps, and optional deformations. Unlike this algorithm, our algorithm does not spend as much time preprocessing each element because we use a more simplified shape descriptor to match each element to its hole.

In contrast to Jigsaw Image Mosaics, Photomosaics deals with creating mosaics from uniformly shaped rectangular tiles on a grid [9]. Tiles are placed based on their color and form. Then, they are color corrected to better represent the target image as a whole. Unlike Photomosaics, we use the physical shape of the element to determine its location in the collage, we and do not alter the elements at all. Simulating Decorative Mosaics builds mosaics out of uniform square tiles [15]. Using Voronoi diagrams to arrange tiles, it ensures the tiles' orientation preserves edges in the target image. By using a signed distance function defined over the target shape, we also keep track of boundaries and use this information to place elements, so they do not protrude beyond the boundary of the target shape. Simulating Decorative Mosaics suggests using smaller tiles in areas of more detail, and we similarly rely on small elements to fill in details of the target shape. Another work, Escherization, takes an arbitrary closed 2D shape and determines a similar shape that can be regularly tiled to fill the space [20]. To create a tileable image shape, it often must deform the input image to some degree. Diverging from Esherization, our method can use a variety of elements in one collage rather than a single tiled element.

II.2. 3D Element Placing Methods

The problem of automatically generating a collage generalizes to the problem of filling a target space with discrete, non-overlapping elements. In the realm of computer science, a number of methods for covering target surfaces or filling target shapes with discrete elements have been developed. Our paper relates most closely to this body of work.

An early work, Cellular Texture Generation, developed a method for synthesizing repetitive discrete elements on a surface [10]. This method simplifies the modeling of surface details like scales, feathers, and thorns. Like our work, their process automatically places discrete elements in a constrained space. However, their process performs a biologically-informed simulation to place its discrete "cells", implemented as particles. Each cell keeps track of a vector of state information including its position and orientation as well as a variety of simulation-related parameters. This algorithm models biological behaviors influencing the cells as first order differential equations which they term "cell programs." Solving this system of equations yields the final state of each cell. The final particle states are converted into the geometry of the discrete elements which can then be rendered. In contrast, our element placement algorithm is a procedural process that places elements in order of their size, filling the target shape from the inside out.

Stereological Techniques for Solid Textures addressed the problem of creating realistic-looking 3D textures for materials with discrete elements embedded within them [18]. Such materials include concrete aggregates, asphalt, igneous rock, as well as sponges which have discrete volumetric voids. Traditional stereological techniques look at 2D slices of 3D materials and draw conclusions from the slices about the volume as a whole. This work presents an approach for using stereological methods to gain statistical information about the density of the discrete elements within the material. Once the density of the elements is known, elements are placed in the volume and simulated annealing is used to resolve collisions. The simulated annealing processes iteratively adjusts the element positions until a certain allowed collision threshold is achieved. Our method, in contrast, places each element so that there is never a collision to be resolved. We do not have the same concern of recreating an existing distribution of elements. For this reason, we have more freedom in placing each element and can guarantee no collisions at all.

Stroke Pattern Analysis and Synthesis presented a method for filling either 1D paths or 2D regions with discrete stroke elements informed by a user-supplied input stroke pattern [3]. From the input pattern, they identify discrete elements, either single strokes or combinations of strokes close in proximity and continuation. They calculate a bounding volume for each element. To synthesize a new pattern, they initialize the target region with a distribution of the elements similar to the input pattern. Then, they iterate through each placed element and analyze its neighborhood of elements to find the most similar neighborhood in the input pattern. They use the properties of the matching input neighborhood to update the synthesized element. In contrast, our method is not influenced by an input example collage. We identify the rules to procedurally generate our collages, and place elements one by one. Once our elements are placed, they are fixed rather than iteratively revisited for adjusting.

Procedural Generation of Rock Piles Using Aperiodic Tiling developed a method for generating piles of rocks in arbitrary target shapes in a way that avoids unrealistic repetitive patterns [26]. They modify the corner cube algorithm to create a set of 256 different cubic tiles. Each tile has a set of points distributed randomly within it. Using these points, they construct a Voronoi diagram with a closed Voronoi cell for each point. Each Voronoi cell corresponds to one rock. To generate the rock geometry from a cell, they randomly choose contact points on each edge and erode the cell away everywhere except at the contact points. This yields a realistic pile of rocks where each touches but does not overlap its neighbors. They ensure consistency across corner cube boundaries by strategically repeating the placement of the Voronoi cell center points in corresponding regions of certain corner cubes. We also try to avoid free-floating elements by orienting each placed element such that one point either touches or is close to touching its nearest neighbor. However, we cannot guarantee that all elements will be contiguous. We rely on a close arrangement of small elements to appear as a contiguous form.

Scales and Scale-like Structures also relies on a Voronoi diagram to place elements in its solution for automatically synthesizing user-defined scales over a surface mesh [22]. A user specifies a direction for the scales with a line to indicate their main orientation. The algorithm generates both a vector field for the scale orientation as well as a Voronoi tessellation for the scale positions over the surface. Using a user-defined scale proxy model, the algorithm merges an instance of the scale onto each Voronoi cell on the model's surface. Unlike this work and others, we do not use a Voronoi diagram to place our elements. Instead, we use procedural rules to place elements, filling the space from the center outward and packing small elements in between.

The use of the Voronoi diagram to place elements within tiles works well for Procedural Generation of Rock Piles Using Aperiodic Tiling [26] as well as for Scales and Scale-like Structures [22] because they generate or modify the element geometry (rocks and scales, respectively) to neatly fill each cell. In our case, using a Voronoi algorithm to divide up the target space then filling each Voronoi cell with a user-defined element would be much more cumbersome. First, the Voronoi cells are more complicated shapes than the spherical holes we consider when placing elements. Thus, using a Voronoi algorithm would necessitate a more sophisticated shape description method for matching our elements to the Voronoi cells. Second, since we place predefined element shapes in holes rather than generating geometry to perfectly fill the holes, there would be unavoidable gaps when an element is smaller than the hole it most closely fits. Further, it would be difficult to place the initial Voronoi sample points to ensure a variety of hole sizes corresponding to the sizes of the user-defined elements.

Piles of Objects describes a physically-based simulation method for building 3D piles with arbitrary elements [16]. Their target shape is a general conical pile or adjoining conical piles. They present a solution for efficiently filling piles of arbitrary angles of repose with large numbers of elements. They simulate the elements falling onto a growing pile of elements with which they collide. Physically-based methods offer the benefit of realistic element arrangements. However, they are usually limited by long simulation times. This work presents a solution to speeding up long simulation times caused by using many simulated elements. They set elements inside the growing pile to "sleep" so that those unseen elements are no longer considered for simulation. This not only speeds the simulation but controls the shape of the pile. Our aesthetic goal differs from that of Piles of Objects in that we do not aim to create physically-plausible stacks of elements. If the target shape of a collage sculpture were a conical pile, the elements would fill it but not look like they were lying at rest. The elements would be close to each other, but they would most likely be touching at corner points instead of contacting on flat edges the way gravity settles elements.

3D Collage: Expressive Non-Realistic Modeling developed a method for filling a user-defined target shape with user-defined element shapes [12]. While our aesthetic goals resemble those of the 3D Collage algorithm, we present a different algorithm to build collages. The following aesthetic goals drive the 3D Collage algorithm: to resemble the target shape, to maximize the visibility of each element, and to restrict overlap and scaling of the elements. Our procedural rules that drive element placement and determine the look of the sculptures differ in that we place less emphasis on the visibility of each element. Further, we completely restrict overlap and more strictly control protrusions beyond the target surface. The 3D Collage method relies on precomputing geometric moments, partial shape descriptors for each element and the target shape, then matching the best fit elements to certain regions of the target. In their work, the $p, q, r-$moment of shape S is: $M_{p,q,r}(S) = \sum x^p y^q z^r$ where the sum is over all surface points. A vector of moments V up to some order d describes each shape. $V(S) = (M_{0,0,0}, M_{1,0,0}, ..., M_{i,j,k}, ..., M_{p,q,r})$, such that $i + j + k \leq d$. They compare an element shape's vector of moments to a target region's vector of moments and find best fit matches. Our method requires much less preprocessing of the elements. We use bounding spheres to categorize our element shapes instead of more sophisticated shape descriptors, and we rely on our constantly updated signed distance function to provide descriptions of open spaces to be filled. We add elements to a collage based on the size of hole remaining to be filled, starting with the largest hole. Thus, the first element will be the biggest and fill the center of the object. Since 3D Collage places more emphasis on the visibility of each object, its elements will be placed mostly on the surface, and its collages will grow in a very different fashion.

In contrast to the procedural collage generation method in both 3D Collage and our own work, Discrete Element Textures presents a data-driven method for synthesizing collages of discrete elements [24]. Users specify an input exemplar, a small combination of elements whose characteristics are mimicked in the output. By allowing users to specify this exemplar, Discrete Element Textures gives the user more control over the output collage than does our procedural method. Our algorithm aims for a more specific aesthetic. In Discrete Element Textures, the user also specifies an output domain, analogous to our target shape, which the algorithm will fill with its collage of discrete elements. They simplify the representation of discrete elements by reducing them to one or more sample points. Each sample records its position and other attributes. We similarly simplify the representation of our elements by using bounding spheres to represent them during placement. Their synthesis algorithm relies on a neighborhood similarity metric that measures the similarity between two patches of elements. Two neighborhoods are similar if they contain many “matching” elements. Elements match if their relative position in the neighborhood and other attributes are similar. They develop an energy formulation to express the desired output collage based on the input exemplar and the output domain, and iteratively minimize this energy formulation to generate a desirable output. Each iteration improves the similarity between each sample’s neighborhood in the output and its most similar neighborhood in the exemplar. Other conditions like restricting elements to lie within the domain shape can also be handled in the energy formulation.

II.3. Sphere Packing

In our process, elements are placed into holes based on the size of their bounding sphere. An element will be centered at a point in the target shape if the radius of its bounding sphere is smaller than the closest surface to that point. To achieve our goal of a space-filling arrangement, it is relevant to look at the sphere packing problem. The sphere packing problem seeks the largest number of non-overlapping spheres that can fit in a target space. Uniform spheres packed in a face-centered cubic arrangement are known to fill about 74% of the volume of their target space [32]. The face-centered cubic packing, a regular packing of uniform spheres, resembles the stacking of oranges at a fruit stand. While the Kepler Conjecture, which states that this arrangement is the most dense packing of spheres, has been widely believed for centuries, it was not proven until a 1998 upper-bound proof by Hales [6]. To achieve densely packed elements, it is important to approximate these dense sphere packings.

II.4. Shape Descriptors

Our method uses bounding spheres as a simple and efficient shape descriptor for matching our elements to their holes. Spheres are efficient to calculate, and they achieved our aesthetic goals for our collage sculptures. However, many more sophisticated shape description methods have been developed. This section explains a number of these shape descriptors. Our method for keeping track of the remaining holes in the target space closely resembles one such method, namely a shape diameter function as developed by Shapira, Shamir, and Cohen-Or [30]. It is described below.

The shape diameter function is a scalar function defined over a mesh that describes the diameter of the object near each point on the surface [30]. It was developed as an expression of volume and thus a consistent descriptor over pose and topology changes for use in mesh partitioning and skeletonisation algorithms. To calculate the shape diameter function, rays are sent in a cone from a surface point in the direction of the inverted normal at that point. The shape diameter value at that point is a weighted average of all the ray lengths for rays that intersected the inside of the mesh. We adapt this idea of a shape diameter function to be defined over a volume instead of over a surface. The idea of a shape diameter function contributed to our representation of the empty volume in a growing collage sculpture as a constantly updated signed distance function. Instead of sending rays to update the distance from a point, though, we use a method to find the exact shortest distance. This way, each point in our target space "knows" the open volume around it.

Many other shape descriptors exist, and a number of them have been compared by testing on the Princeton Shape Benchmark [31]. The Princeton Shape Benchmark is a large database of polygonal models with categorized testing and training sets for comparing different shape description and matching algorithms.

Modeling by Example uses a shape descriptor for performing both partial and whole shape matching in an interactive modeling application [11]. In their application, a user creates a model by selecting components from similar existing models in a database. Components are classified as similar via similar shape signatures. For each model A in the database, there are two voxel-based representations of the model: the first is a rasterization of the boundary with a value of 1 in voxels that intersect the boundary and 0 otherwise, R_A; the second is a squared Euclidean Distance Transform of the boundary, E_A. To compare two shapes, the voxel representations are used to capture the distance between corresponding points on the two shapes. Those with a smaller overall difference are more similar. The distance between two shapes A and B is computed as follows: $d(A, B) = R_A \bullet E_B + E_A \bullet R_B$.

Fully Automatic Registration of 3D Data Sets uses shape descriptions and partial shape matching to reconstruct a 3D shape by piecing together portions of the model created by 3D scans from different but overlapping viewpoints [17]. It relies on visibility consistency or the similarity of two surfaces along the line of sight from two different viewpoints to match surfaces.

Multi-scale Features for Approximate Alignment of Point-based Surfaces introduces a method for extracting salient points from 3D scan data [23]. Since 3D scanner data is so complex, it is important to have a smaller subset of reliable points to compare between shapes. Then, shape descriptions can be computed using these points to compare two partial shapes.

Salient Geometric Features for Partial Shape Matching and Similarity describes a means of calculating a small number of salient features on a triangulated mesh to enable partial matching of meshes. It uses a hierarchical approach as it first calculates local descriptors to represent patches of the surface. Then salient features are calculated to represent groups of the local descriptors. Curvature of the surface is one of the main factors in determining features. Areas of high curvature have more descriptors and thus features. These descriptors are precomputed for a database of objects, then geometric hashing is used to find matches [13].

While we chose bounding spheres as a simplified shape descriptor, a large number of more sophisticated shape descriptors exist. Chapter VI suggests possible future work that could benefit from using a different shape descriptor in our Collage Sculpture process.

CHAPTER III

PROCESS

III.1. Process Overview

Here we present the three steps in our process for creating a collage sculpture.

1. *Preprocessing.* To enable element placement, this step calculates a signed distance function for the user-defined target shape and calculates the radius of an enclosing bounding sphere for each user-defined element shape. Section III.2 provides further description.

2. *Element Placement.* To assemble an arrangement of elements into a collage sculpture, this step selects and positions each element in the target shape. This represents the main body of the thesis work. Section III.3 provides further description. This step consists of two procedures.

   (a) *Placement.* This procedure takes care of selecting, placing, and orienting an element in the target space.

   (b) *Updating.* In response to the placement of each element, this procedure updates the signed distance function for the next placement.

3. *Export and Render.* To save and view the final collage sculpture, this step records the arrangement of elements and creates an image of the result. Section III.4 provides further description.

III.2. Preprocessing

The first step of the collage sculpture process is preprocessing the user-defined target and element shapes. Here we prepare the user-defined shapes for use in the *Element Placement* step.

To place elements in the target shape, we need to know the distance to the nearest surface from each point. This way, we know the size of element we can place at each point to avoid overlapping the nearest surface. To keep track of these distances, we define a signed distance function over a volume enclosing the target shape. This makes it possible to keep track of empty space waiting to be filled by elements. The signed distance function value at a given point is equal to the size of element that could be centered at that point.

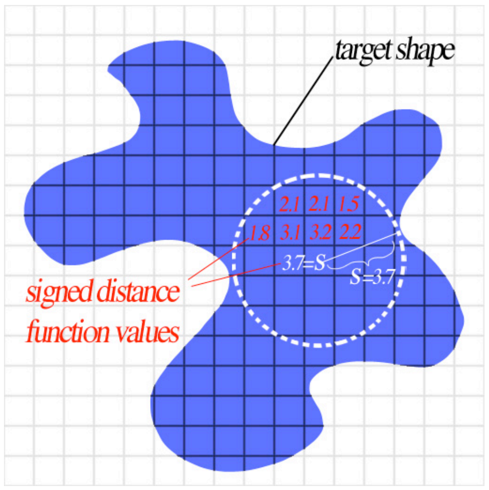


Fig. 4. Signed distance function.

Fig. 4 shows a hypothetical 2D example of a signed distance function defined over a target shape. Example signed distance function values are depicted in red and white. At any point with a signed distance function value S, there is a sphere of open space with radius S. An element of size S centered at that position will not overlap another surface. For our needs, it is enough that the signed distance function be defined at discrete points over the target volume. We store one signed distance value per voxel corresponding to a point within that voxel. This point is either the exact center of the voxel, or, optionally, a point jittered from the voxel's center to break up the grid-based regularity of the voxel samples. If the voxel's sample point is jittered, all signed distance values are calculated from that offset sample point to ensure accuracy.

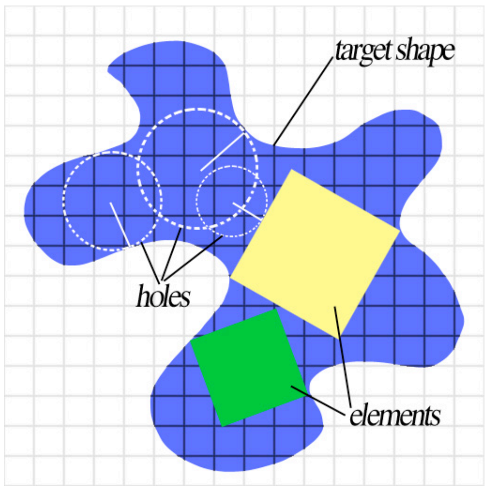


Fig. 5. "Holes" in target space.

Unless a point is filled by an element, it correlates to a “hole” in the target space. A “hole” in the target space is a sphere of empty space. The radius of the hole is the value of the signed distance function at the hole’s center point. Fig. 5 identifies three holes in this particular target space. The radius of each is the distance from its center to the nearest surface. In other words, the radius of each is the signed distance function value at its center.

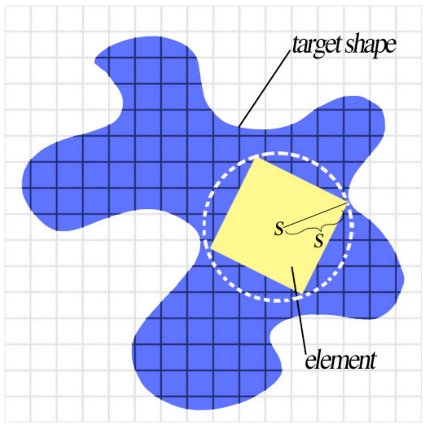


Fig. 6. Element placed by “size.”

To select an element to fill a hole in the target shape, we need to know the “size” of the element. The “size” of an element is the radius of its smallest enclosing bounding sphere. Given an element’s size, we know which holes we can fill with it. An element will fit in a hole whose radius is greater than or equal to the element’s size. Fig. 6 illustrates an element with size S fitting exactly in a hole of radius S.

Thus, in *Preprocessing*, we calculate the radius of the smallest enclosing bounding sphere for each element as well as the target space signed distance function.

III.3. Element Placement

Once we have the size of each element and the target space signed distance function to understand the holes, we can place elements. We identified four aesthetic rules to govern the *Element Placement* step. Each element will be placed in such a way to abide by all rules.

- *Visually clarify the target shape.* The element arrangement must recognizably represent the target shape. To this end, our elements will adequately fill the target space and not protrude beyond its boundaries.
- *Avoid element overlap.* The arranged elements must not overlap each other at any point.
- *Avoid element deformation.* The elements must not be deformed to fill a hole in the target space. They must be used as found.
- *Increase variety.* The element arrangement must display variety in element size, orientation, and position.

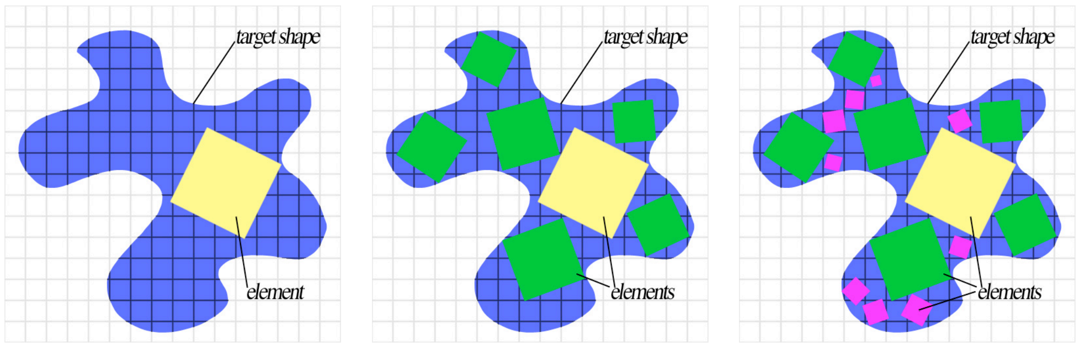
Using these rules to guide *Element Placement* guarantees collage sculptures that meet our desired aesthetic. *Element Placement* consists of two procedures.

1. *Placement.* This procedure selects holes in a specific order and chooses the elements to fill those holes. It places elements in a specific orientation.
2. *Updating.* This procedure updates the target space signed distance function to account for each newly added element. After updating, the signed distance function accurately represents the remaining open space.

III.3.1. Placement

To incorporate the largest variety of element sizes, we place elements starting with the largest that fits in the target space and continuing with increasingly smaller elements. Once an element is placed, it fragments the remaining space so that progressively smaller spaces are available. By placing elements biggest to smallest, we are able to put big elements in large blocks of contiguous space. Smaller elements fill in the remaining space. Fig. 7 depicts our element placement from largest to smallest. We stop placing elements when the largest hole left is smaller than a certain cutoff size.

Fig. 8 illustrates the space fragmentation that can occur if elements are added in the reverse order of smallest to largest. Contiguous blocks of space are divided so that larger elements no longer fit.



(a) First placement (b) Next placements (c) Last placements

Fig. 7. Elements added in order of large to small allow a variety of element sizes.

To accomplish placing elements from largest to smallest, we choose holes in order from biggest to smallest and fill each with the largest element that fits. Now that we have the target space signed distance function, we can locate the largest hole. This corresponds to the point with the greatest signed distance function value. Since we have preprocessed each element to determine its largest enclosing bounding sphere, we can select the element with the largest size less than or equal to the radius of the hole. This way, no element overlaps an already-placed element. Similarly, no element protrudes beyond the boundary of the target shape.

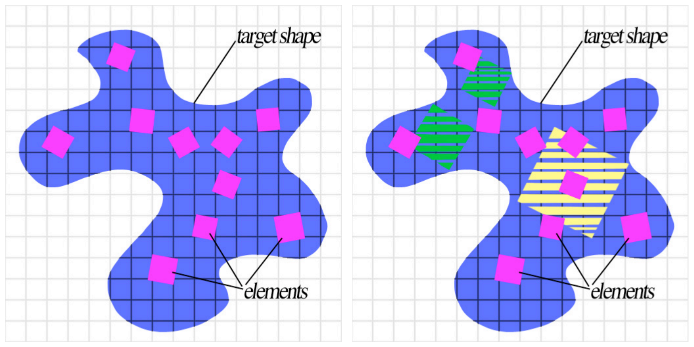


Fig. 8. Space fragmentation if small elements are placed first. Larger elements cannot be placed.

To incorporate a variety of element orientations, we orient each element in its hole so that one point of it just touches, or comes close to touching, its nearest neighboring surface when it is placed. If all elements were exactly as large as the hole in which they were placed, all elements would be touching at least one neighbor. Without scaling the user elements, however, this will not always be the case. Nonetheless, this procedure ensures a variety of orientations. Also, it discourages “floating” elements that break up the continuity of the target shape.

III.3.2. Updating

For the signed distance function to accurately represent the remaining holes, we must recalculate it after each element is placed. This way, each subsequent element is guaranteed to not overlap a previous one.

To have an accurate measure of space between and among each newly placed element, we update the definition of the “surface” to which the signed distance function measures. Initially, the signed distance function measures the distance from each point to the target shape surface. With each new element, “surface” is expanded to include the new element’s surface as well. We update the signed distance function at those points whose new closest surface is the new element. This occurs for each new element. This way, the signed distance function accounts for filled holes. It constantly represents the open space left in the growing collage sculpture.

To eliminate filled positions from consideration for later placement, the *Updating* procedure begins by flagging all points that lie inside a new element. After flagging interior points, it proceeds to update the signed distance function for points whose new closest surface is the new element. So, we loop through each position and determine if the new element now brings the surface closer than the signed distance function previously recorded. We can quickly eliminate points that do not need updating by calculating the distance from each point to the new element’s bounding sphere. If the distance to the bounding sphere is greater than or equal to the point’s current signed distance function value, that point is too far from the new surface to need updating. Then, for each position not eliminated by the bounding sphere distance test, we find the nearest point on the new element surface. If the distance to this point is smaller than the position’s current signed distance function, we update that signed distance function value.

Placement and *Updating* repeat in turn until the largest hole left is smaller than a certain cutoff size. Once that cutoff size is reached, we proceed to the *Export and Render* step.

III.4. Export and Render

In this step, we record all the placed elements together with their positions and orientations in a format we can use to render the final collage sculpture. We render the final collage sculpture, and the process is complete.

CHAPTER IV

ELEMENT PLACEMENT ALGORITHM

The *Element Placement* step in the collage sculptures process is responsible for selecting and placing the elements in a collage sculpture. It consists of two procedures: (1) *Placement*, (2) *Updating*. The structure of the algorithm is shown in Fig. 9. It is detailed in this chapter.

```
preprocessing;
$r$; {maximum radius of the bounding sphere of next element}
$p$; {position of the center of next element}
$o$; {orientation of next element}
$r$, $p$, $o$ = find voxel with largest signed distance function;
while $r \geq threshold$ do
    find element with closest radius to $r$, such that radius $\leq r$;
    position element at $p$;
    orient element with $o$;
    updating;
    $r$, $p$, $o$ = find voxel with largest signed distance function;
end while
export;
```

Fig. 9. Main collage sculpture process algorithm.

IV.1. Placement

After preprocessing, the program enters the main loop of the *Element Placement* algorithm which places an element on each iteration. To place elements in order from biggest to smallest, it looks for the largest remaining hole to fill. The largest remaining hole is centered at the voxel with the largest signed distance function value. Thus, the program continues to place elements until the largest remaining signed distance function value is smaller than the cutoff, *threshold*. The choice of *threshold* affects the look of the collage sculpture: too large, and details remain unrepresented; too small, and every little hole is filled in an extremely dense collage.

For the largest signed distance function value found on each iteration, there is a hole to be filled. The program selects the element to place based on its bounding sphere radius. It centers the element at the voxel's effective center, filling the hole.

To orient the element, the program aligns the element's stored direction from its center to farthest point with the voxel's direction to nearest neighbor. This way, at least one point on the new element touches, or reaches near, its nearest neighbor when it is placed.

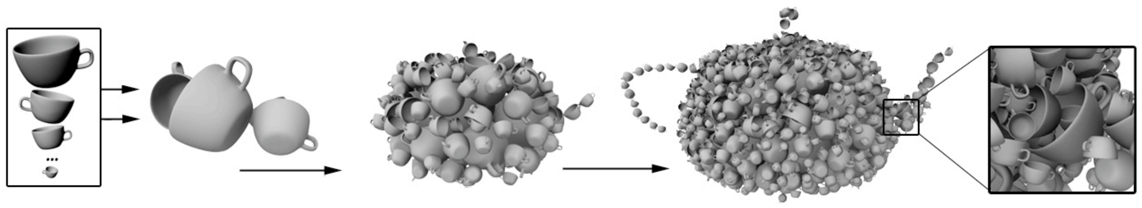


Fig. 10. Assembling a collage sculpture. Teacup elements placed in teapot target.

Fig. 10 captures a collage sculpture at three stages during its assembly. The elements consist of teacups in a wide range of sizes while the target is a teapot. First the largest teacups are placed in the center. Gradually smaller and smaller elements are placed with no intersections. No more teacups are added when the remaining holes are smaller than the *threshold*. If the *threshold* were larger, the sculpture might look like either of the first two stages. Fig. 11 shows the resulting collage sculpture with the target shape overlaid. No teacups protrude beyond the target boundaries.

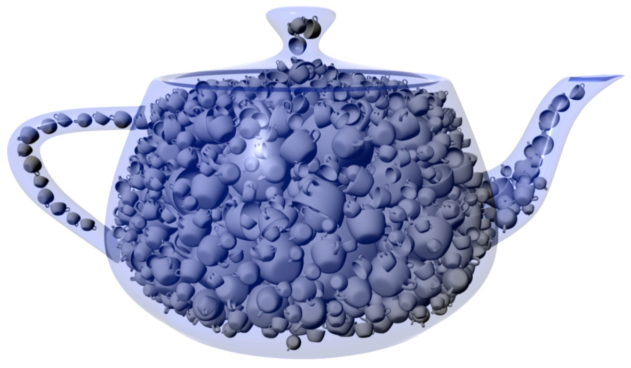


Fig. 11. Collage sculpture within target boundaries.

IV.2. Updating

The *Updating* procedure addresses and recomputes the signed distance function at each voxel that might be affected by a newly placed element. Fig. 12 outlines the structure of this algorithm.

```
flag interior voxels;
for v in all voxels do
  if v is not inside an element then
    d = distance from v to new element's bounding sphere;
    if d < (v's signed distance function value) then
      calculate closest distance from v to new element;
      update v's signed distance function value;
      update v's closest direction vector;
    end if
  end if
end for
```

Fig. 12. Updating procedure algorithm.

IV.2.1. Flag Interior Voxels

Updating begins by setting the signed distance function to zero at each voxel that lies inside the new element. It loops through voxels in a bounding volume around the new element. For each voxel it performs the classic inside/outside test.

1. *Send a ray in a given direction.*
2. *Count the number of faces it hits.*
3. *Even number* $\Rightarrow$ *outside; odd number* $\Rightarrow$ *inside*

To reduce error arising from a ray hitting the edge of two faces, we sent two rays.

- First ray: sent in the direction of one face's normal.
- Second ray: sent in a random direction.

We ensure the number of hits for each ray is the same. If not, we take the least number of hits because rays that intersect the edge of multiple faces erroneously add all intersected faces to the hit count. For our application, this test performed adequately.

IV.2.2. Compute Closest Surface Point

Once the interior voxels are flagged, all voxels whose closest surface point is now the new element need updating. To quickly eliminate voxels too far from the new element to need updating, we compute the shortest distance from each voxel to the new element's bounding sphere.

1. *Calculate shortest distance, d, from voxel to bounding sphere:*
   $d = |p_e - p_v| - r$,
   p_e: *position of the element*
   p_v: *position of the voxel*
   r: *bounding sphere radius*
2. *If d <(voxel's signed distance function)* $\Rightarrow$ *voxel may need updating*
3. *Else* $\Rightarrow$ *voxel doesn't need updating*

For each voxel that may still need updating, we compute the closest point on the new element surface. If the distance to that point is less than the voxel's current signed distance function, it is updated. To compute the closest point, we search for the closest point among all the vertices, edges, and faces of the polygonal element.

The closest among these is the overall closest point to the voxel. Refer to Fig. 13 for the structure of this algorithm. It is important that each face is a plane, and for this reason, we require the element OBJ shapes to be triangulated.

To find the closest vertex to a given voxel, we calculate the squared distance to each vertex. The equation to find the squared distance D between points $p_1 = (x_1, y_1, z_1)$ and $p_2 = (x_2, y_2, z_2)$ is below.

$$D(p_1, p_2) = (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2 \tag{4.1}$$

Continuing with the computation of the closest surface point on an element, we check each of the edges and faces for a closer point. For each edge (face), we calculate the closest point p on the infinite line (plane) on which that edge (face) lies. If p is within the edge (face), it is a candidate for the closest surface point. If it is a candidate, we compute D from the voxel to p. If D is less than the currently saved best D, we update the best D. Figs. 14 and 15 show the edge and face cases.

Below are the equations for calculating the closest point p on an edge.

- Let p_1 and p_2 be the vertices of the edge.
- Let p_3 be the voxel point from which we are calculating.

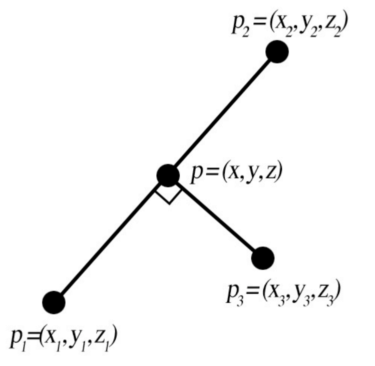
$$p = p_1 + u(p_2 - p_1) \tag{4.2}$$

$$(p_3 - p) \bullet (p_2 - p_1) = 0 \tag{4.3}$$

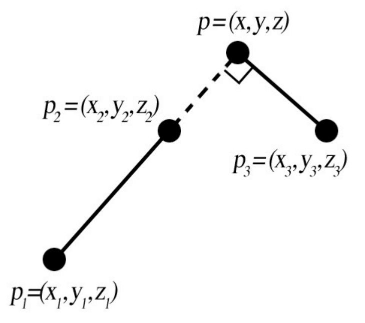
Equation 4.2 defines the infinite line on which the edge lies. u is a parameter along that line. u is 0 at p_1 and 1 at p_2, so if $u \in [0, 1]$ then p is in the line segment between p_1 and p_2. Equation 4.3 states that $p_3 - p$, is perpendicular to $p_2 - p_1$. This holds because the closest direction from a point to a line (p_3 - p) is always perpendicular to the line direction (p_2 - p_1). To calculate p on an edge:

```
pt; {point from which to measure shortest distance}
e; {polygonal element}
d = MAX FLOAT; {shortest distance from pt to e}
for vert in vertices of e do {compute distance to vertices}
    tempD = squared distance from pt to vert;
    if tempD < d then
        d = tempD;
    end if
end for
for edge in edges of e do {compute closest distance to edges}
    p = closest point on line to pt;
    if p in edge then
        tempD = squared distance from pt to p;
        if tempD < d then
            d = tempD;
        end if
    end if
end for
for face in faces of e do {compute closest distance to faces}
    p = closest point on face to pt;
    if p in face then
        tempD = squared distance from pt to p;
        if tempD < d then
            d = tempD;
        end if
    end if
    d = $\sqrt{d}$;
end for
```

Fig. 13. Algorithm to compute shortest distance from a point to an element's surface.

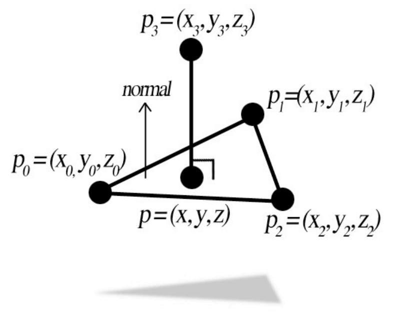


(a) Closest point within edge

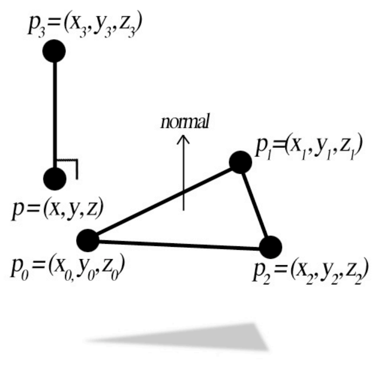


(b) Closest point outside edge

Fig. 14. Two cases for the shortest distance between a point and an edge.



(a) Closest point within face



(b) Closest point outside face

Fig. 15. Two cases for the shortest distance between a point and a face.

1. *Solve for u.* Substitute Equation 4.2 into Equation 4.3.

$$u = \frac{(p_3 - p_1) \bullet (p_2 - p_1)}{(p_2 - p_1) \bullet (p_2 - p_1)} \tag{4.4}$$

2. *If $u \ni [0,1] \Rightarrow p$ not within the edge. Go to next edge.*

3. *Else $\Rightarrow p$ within edge, calculate D.*

4. *If $D <$ (previous best D) $\Rightarrow$ update best D.*

Once the closest point among vertices and edges is found, continue to faces. Below are the equations for calculating the closest point p on a face.

- Let p_0, p_1, and p_2 be the vertices of the face.

- Let p_3 be the voxel point from which we are calculating.

$$n \bullet (p - p_0) = 0 \tag{4.5}$$

$$p = p_3 + n_3 u \tag{4.6}$$

Equation 4.5 is the equation for a plane with normal n in which the vertex p_0 lies. This is the plane in which our triangular face lies. Equation 4.6 is the definition of a ray that begins at point p_3 and travels in the direction of n_3. u is a parameter along that ray. The point on a plane that is closest to p_3 will be a projection of p_3 onto the plane in the opposite direction of the plane's normal. Thus, $n_3 = n$. In other words, the direction from p_3 in which we are looking for an intersection on the face is the negative of the face's normal. If u is positive, the point along the ray it describes is in the direction of n_3 from p_3. If u is negative, the point is behind p_3. To calculate p on a face:

1. *Solve for u.* Substitute Equation 4.6 into Equation 4.5.

$$u = \frac{n \bullet (p_0 - p_3)}{n \bullet n_3} \tag{4.7}$$

2. *If $u < 0 \Rightarrow$ plane is parallel to ray or plane is behind ray. Go to next face.*

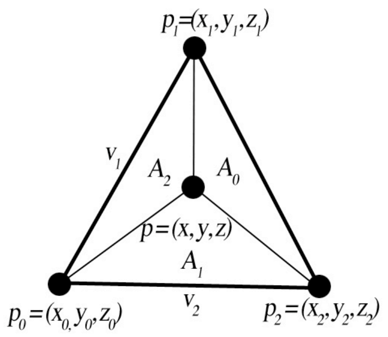
3. *Else $\Rightarrow$ calculate p.*

4. *Determine if p is within the face.*

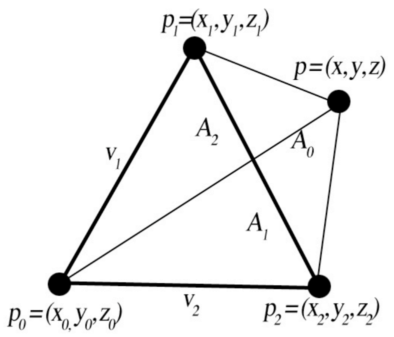
5. *If p is within the face $\Rightarrow$ calculate D.*

6. *Else $\Rightarrow$ Go to next face.*

7. *If D < (previous best D) $\Rightarrow$ update best D.*



(a) Closest point within face



(b) Closest point outside face

Fig. 16. Determining if point on a plane is within a triangular face.

To determine if p lies within the triangular face formed by p_0, p_1, and p_2, we look at the relationship between the areas of the triangles formed by each of the edges and p. Refer to Fig. 16 for illustration. v_1 and v_2 are the vectors defining two edges of the face.

$$v_1 = p_1 - p_0 \tag{4.8}$$

$$v_2 = p_2 - p_0 \tag{4.9}$$

The area of a triangle is the half the magnitude of the cross product of the edge vectors.

$$A = \frac{1}{2}|v_1 \times v_2| \tag{4.10}$$

We define an area vector, $\vec{A}$ for the triangle face that is the cross product of v_1 and v_2.

$$\vec{A} = v_1 \times v_2 \tag{4.11}$$

Likewise, we define area vectors for the triangles formed by each of the edges and p, the intersected point in the plane.

$$\vec{A_0} = (p_1 - p) \times (p_2 - p) \tag{4.12}$$

$$\vec{A_1} = (p_2 - p) \times (p_0 - p) \tag{4.13}$$

$$\vec{A_2} = (p_0 - p) \times (p_1 - p) \tag{4.14}$$

To determine if p lies inside the face, we compare the direction of the area vectors $\vec{A}$, $\vec{A_0}$, $\vec{A_1}$, $\vec{A_2}$. If they all face the same direction, p is within the face. Otherwise, p is outside the face. We can test that all area vectors face in the same direction by comparing the signs of the same non-zero component of each. If the signs are all matching, p is inside the face. Then, we use Equation 4.1 to calculate the squared distance from p_3 to p. If this squared distance is shorter than the previous saved shortest distance to the element, we replace the shortest distance. After we have looped through all vertices, edges, and faces of an element, we know the shortest squared distance from the voxel to the element.

Finally, given the shortest squared distance from a voxel to the element, we take the square root to get the absolute shortest distance d. If d is less than the voxel's signed distance function value, we update it.

To summarize, the *Element Placement* algorithm fills holes in the target space from largest to smallest updating the target space signed distance function after each placement. When updating the signed distance function, the algorithm first flags interior voxels using a variation of the classic inside test. Then, it updates the signed distance function values for voxels near the new element. A voxel is considered "near" the new element based on its distance to the element's bounding sphere. For all near voxels, the exact distance to the new element is calculated by finding the shortest squared distance among the distances to the element's vertices, edges, and faces. The computed shortest distance is compared to the voxel's signed distance function value. If the new distance is smaller, the signed distance function value is updated.

CHAPTER V

IMPLEMENTATION AND RESULTS

For this thesis, I developed a C++ program and procedures for interacting with both Houdini 10 and Maya, two 3D animation packages, to implement the collage sculpture process. I used Houdini in the *Preprocessing* and Maya in the *Export and Render* steps of the process. I implemented the *Element Placement* step completely using C++. In this chapter, I describe the implementation of each of the three process steps: (1) *Preprocessing*, (2) *Element Placement*, (3) *Export and Render*.

V.1. Preprocessing

My preprocessing implementation combines Houdini and C++. I leverage Houdini's exisiting tools to compute the initial signed distance function for the target shape. Using Python within Houdini, I output these values to a text file, and load them into my C++ program which initializes data structures to handle both the target and element shapes.

V.1.1. Houdini Signed Distance Function Calculation

To compute the initial signed distance function for the target shape, I import the user-specified target shape into Houdini. I use triangulated OBJ format files to specify the target and element shapes. In Houdini, an isoOffset node computes the signed distance function values. Refer to Fig. 17 for the node network. The upper left (green) node represents the OBJ mesh for the target shape. The upper right (blue) node represents the rectangular bounding volume to define the target space. The bottom (yellow) node is the isoOffset node which calculates the signed distance function. In this node, I set the sample size for the target space which determines the discretization of the sample space. Houdini discretizes the bounding volume space into cubic voxels and calculates a signed distance function value at the center of each voxel. Using Python within Houdini, I output a text file with the dimensions of the voxel space and the signed distance function value for each voxel. I then load this text file into my C++ program.

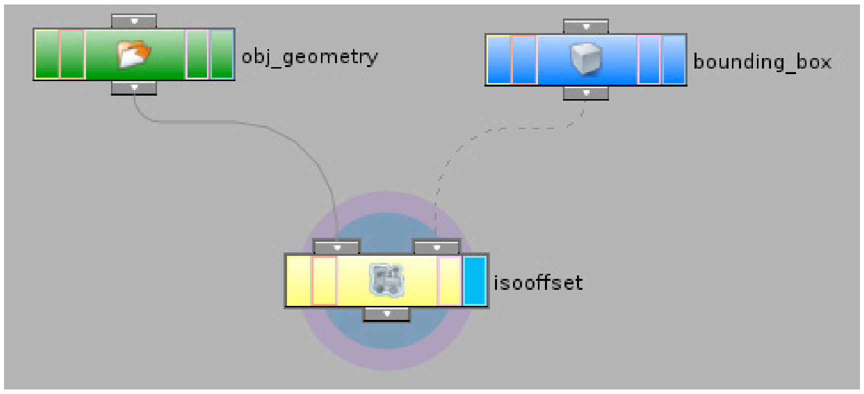


Fig. 17. Houdini node network for computing target shape's signed distance function.

V.1.2. C++ Preprocessing and Data Structures

At startup, the C++ program takes both a text file with the target shape signed distance function values and a collection of OBJ files for the element shapes. The element shapes are triangulated polygonal meshes.

The preprocessing for the target space uses Houdini's discretization of the target space, and creates a cubic voxel for each signed distance function sample point. Each voxel stores its signed distance function value, its position in the voxel space, a direction vector to the closest surface, and a jitter vector which defines an offset from the actual center of the voxel to a jittered center point at which the signed distance function will be defined. By jittering the voxel center points, we break up the grid-based regularity of the voxel structure, so elements are placed with more positional variety.

V.2. Element Placement, Export and Render

After preprocessing, the C++ program proceeds directly to the element placement main loop. This is described in detail in Chapter III.3. The program arranges the elements in the target shape. Then, a user can interactively view the generated collage sculpture. I used OpenGL to draw the elements to a screen. This aided development and debugging. To export the final sculptures, I wrote out all the arranged elements as a single OBJ file.

To render the OBJ files, I imported them into Maya and set up an environment image for performing final gather. I used Mental Ray to render the final images.

V.3. Results

We created several collage sculptures to demonstrate the capabilities of this program, and they are shown in Figs. 18, 19, 20, 21, and 22. The C++ program was developed and tested on a 2.4 GHz Intel Core 2 Duo computer. The times to assemble the collage sculptures shown here ranged between less than a minute to about 45 minutes. Table I compares the specifications of each collage sculpture.

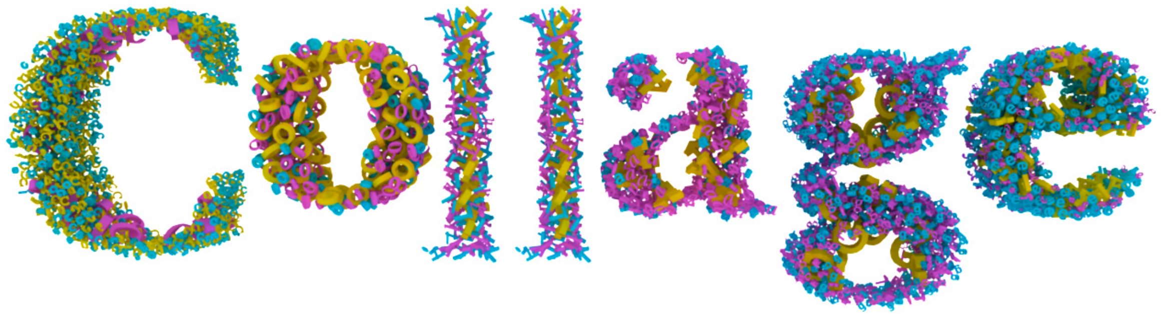


Fig. 18. “Collage” collage sculpture.

In Fig. 18, each letter of the word “Collage” is an individual collage sculpture. The elements comprising each letter are models of that letter in three different fonts. There are three unique elements in each collage sculpture letter. For example in the letter “C”, the yellow elements are small 3D models of a “c” in a first font, the magenta elements a second, and the blue a third.

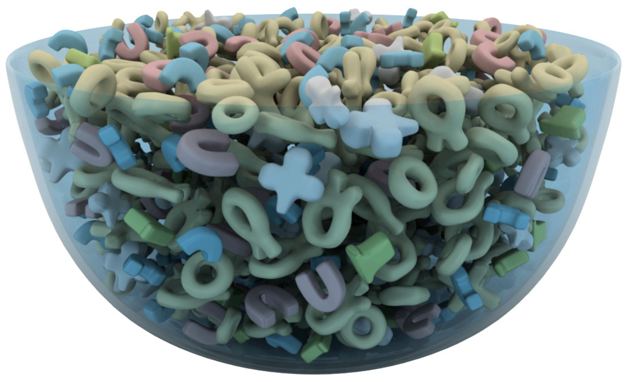
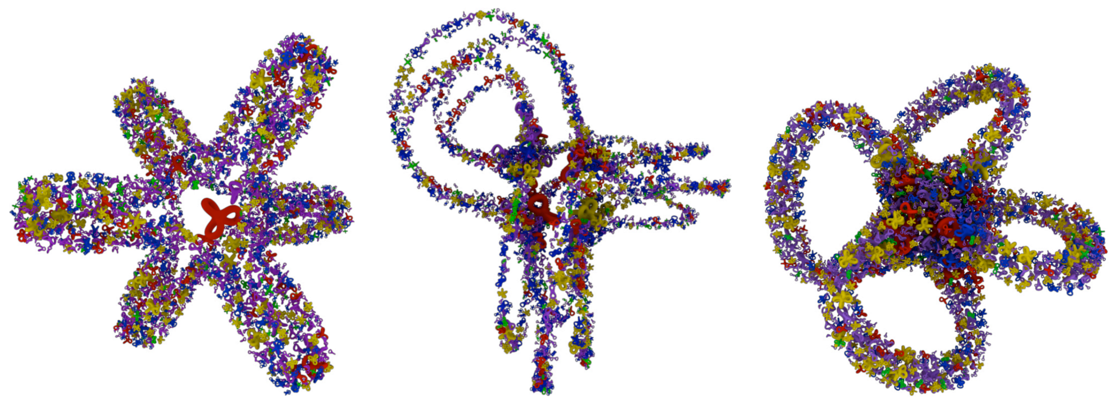


Fig. 19. Cereal collage sculpture.

To make the cereal collage sculpture in Fig. 19, I modeled a variety of different cereal pieces as well as the target shape that resembles the inside of the bowl. Since the program uses the elements without modifying them, relative scale is an important consideration in modeling the elements and target shape. A collage sculpture like this might be a good way to initialize elements for a physically-based simulation. See Chapter VI for a further discussion of this as future work.



(a) TopMod1 (b) TopMod2 (c) TopMod3

Fig. 20. Collage sculptures with TopMod shapes.

Fig. 20 shows three different collage sculptures, each modeled using target and element shapes created by the program TopMod (Topological Mesh Modeling). This illustrates how both the element and target shapes can be arbitrary models built from any modeling application.

With the chair, teapot, and bunny sculptures (Figs. 21 and 22), we aimed to fill the target space as densely as the algorithm would allow. To this end, we relaxed the procedural rule forbidding element deformation and allowed scaling of the elements in these cases. To do this without deforming the elements, theoretically, we would

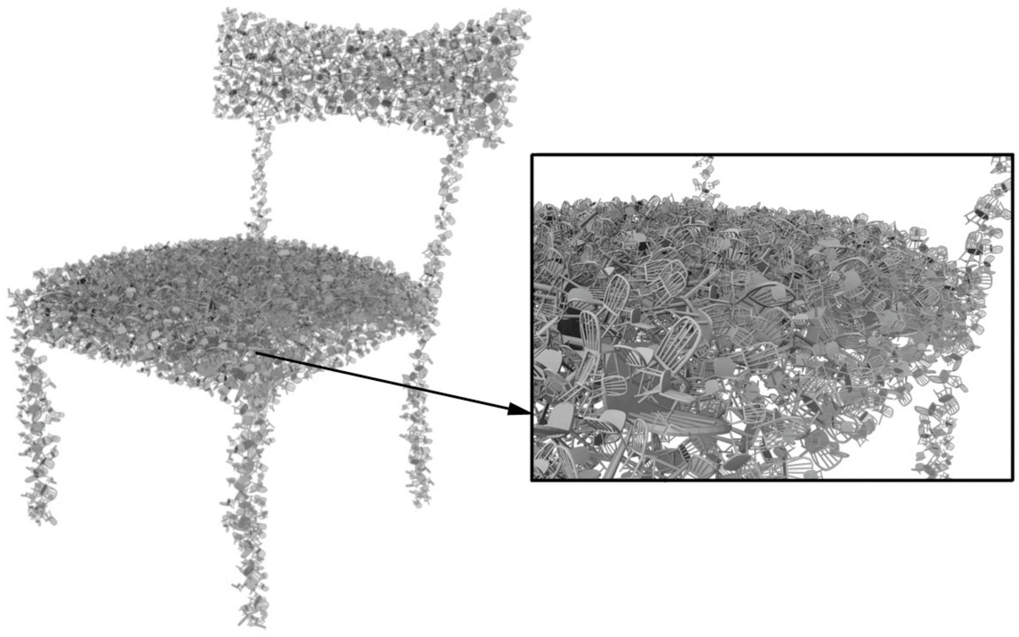
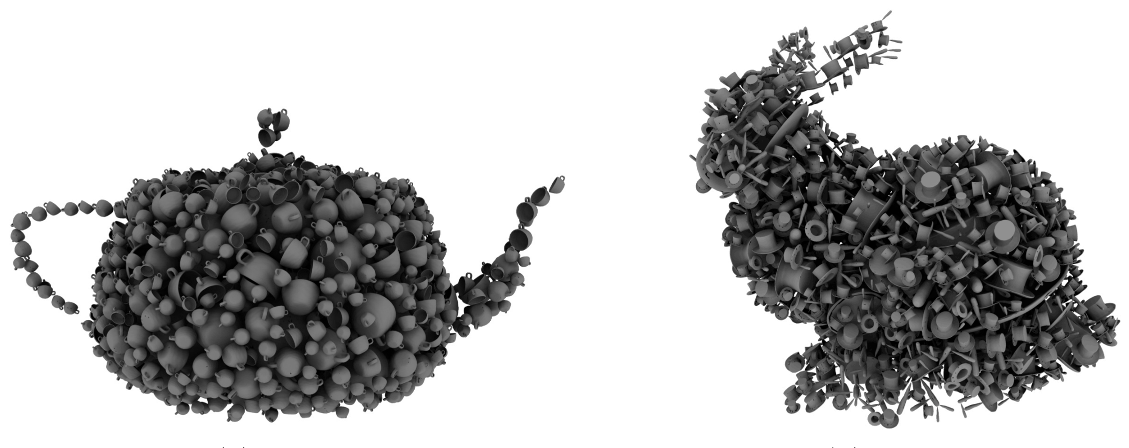


Fig. 21. Chair collage sculpture.



(a) Teapot

(b) Bunny

Fig. 22. Teapot and Bunny collage sculptures.

input a collection of elements in an infinite range of scales. However, to practically implement this, we input just the unique element shapes and scaled the them during the *Placement* procedure so that their size exactly aligned with that of the hole in which they needed to be placed. The chair elements in the chair collage sculpture are scaled versions of a model taken from the Princeton Shape Benchmark database [31]. I modeled the target chair. The Utah teapot is composed of teacups while the Stanford bunny is composed of a combination of carrots and top hats. I modeled their elements. For the bunny, composed of two unique shapes, we alternated between placing the two shapes (hat, carrot, hat, carrot, etc.). This way, we achieved a nearly even distribution of each element shape. Fig. 23 distinguishes the two element shapes in the bunny sculpture.

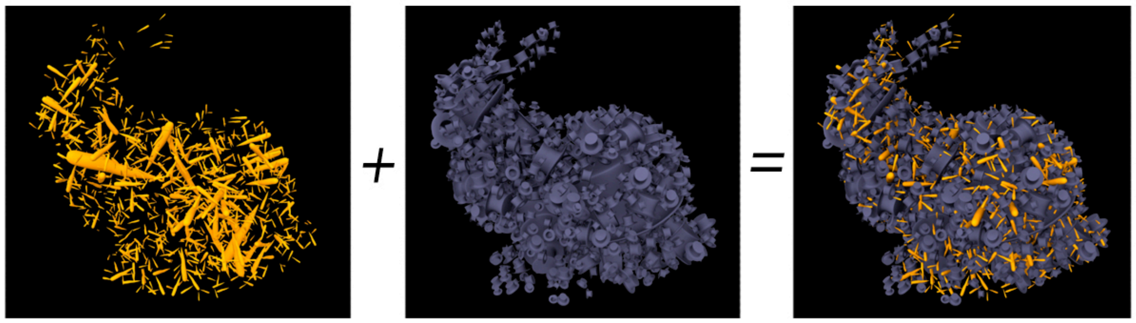


Fig. 23. Bunny sculpture breakdown.

Due to our choice of using bounding spheres as the element shape descriptors, elements whose form is more similar to a sphere pack the target space more densely than elements whose form diverges from a sphere. Note how the rounded teacups pack closely into their teapot target. By comparison, note the sparser packing of the carrots in the bunny's ears. Lowering the threshold value would increase the carrot density. However, if the same threshold were used in packing a target shape with nearly spherical elements versus long thin elements, the spherical elements would better fill out the target. Fig. 23 makes this very apparent. Notice it is much harder to distinguish the bunny shape from the carrot arrangement than it is from the top hat arrangement though the sizes and number of carrot elements corresponds nearly equally to the top hat elements.

Table I. Result Comparisons

Collage	Voxel Dimensions	Total Elements	Faces Per Element	Assembly Time
Teapot	100×100×100	1433	300	10.00 min
Bunny	100×100×100	2049	208, 288	18.08 min
Cereal	100×55×100	565	108 – 248	12.25 min
Chair	100×100×100	3499	1144	44.88 min
TopMod1	100×100×100	2614	72 – 480	2.92 min
TopMod2	100×100×100	2303	72 – 480	2.23 min
TopMod3	100×100×100	2905	72 – 480	5.75 min
Collage–C	100×100×50	1584	156, 172, 236	3.62 min
Collage–O	100×100×50	452	128, 128, 144	1.23 min
Collage–L	100×100×50	323	12, 52, 188	0.87 min
Collage–A	100×100×50	585	84, 304, 320	2.55 min
Collage–G	100×100×50	1574	304, 348, 452	8.20 min
Collage–E	100×100×50	1128	92, 192, 224	2.15 min

The specifications for each collage sculpture are compared in Table I. In each case, we used cubic voxels with 100 voxels along the longest dimension of the target shape. A finer grid would create collage sculptures that better represent the finer details of the target shape. However, increasing the number of voxels increases the number of holes searched in the *Placement* procedure and the number of voxels updated in the *Updating* procedure. The assembly times represent the runtime for the *Element Placement* step of the collage sculpture process. Thus, they account for the time taken for every placement and signed distance function update in the assembly of a collage sculpture. On average for our collage sculptures, the *Updating* procedure took 93.13% of the listed assembly time. *Placement* accounted for the other 6.87%. Increasing the number of faces per element leads to more detailed element shapes, but it also increases the updating time. Nonetheless, our method for calculating the exact distance to each element during *Updating* is much more efficient than a ray tracing alternative where ray intersections are used to sample distances to the element.

Since *Updating* is the most expensive procedure, we considered the speedup we could achieve by abbreviating the computation of the closest surface point to only the closest vertex search. In doing this, we update signed distance function values to reflect the closest vertex and ignore the edges and faces. This is an approximation that will lead to some overlap among elements. For element models with a large number of vertices, this approximation is reasonable. However, for elements with large faces, like a cube whose vertices only lie at its eight corner points, there will be noticeable overlap. Nonetheless, we computed the time to assemble each collage sculpture with this approximate updating method to compare the assembly times. We used the same cutoff values for each sculpture, but since the elements overlap with the vertices-only method, more elements were placed in each of these sculptures. This made it difficult to directly compare the assembly time. However, when we looked at the assembly time per element for each of the collage sculptures, the vertices-only method gives an average speedup of 1.32. Additionally, when using the vertices-only method, on average, the *Updating* procedure took 90.04% of the assembly time with *Placement* taking the remaining 9.96%.

Further, we analyzed the density to which our placed elements fill the target volume. The collage sculpture process does not aim to achieve an optimally dense packing, but it is interesting to consider the density of elements required to construct a recognizable target shape. We calculated the density of each collage sculpture by dividing the number of voxels flagged as interior to the elements by the number of voxels inside the target shape. The results are displayed in Table II. Again, elements that more closely approximate their bounding sphere better fill their target space. The Chair collage sculpture with chair elements that have long, thin legs and backs is only 1.44% filled. Similarly, the “l” in the “Collage” collage sculpture also consists of very long and thin elements that fill just 7.39% of the target. To contrast with spherical elements, we filled the teapot target shape with spheres (see Fig. 24). The spheres fill 72.87% of the target volume.

Table II. Density Comparisons

Collage	Percent Filled
Teapot–teacups	11.04
Bunny	10.90
Cereal	14.34
Chair	1.44
TopMod1	13.82
TopMod2	13.50
TopMod3	14.73
Collage–C	9.28
Collage–O	14.33
Collage–L	7.39
Collage–A	14.55
Collage–G	8.84
Collage–E	16.71
Teapot–spheres	72.87

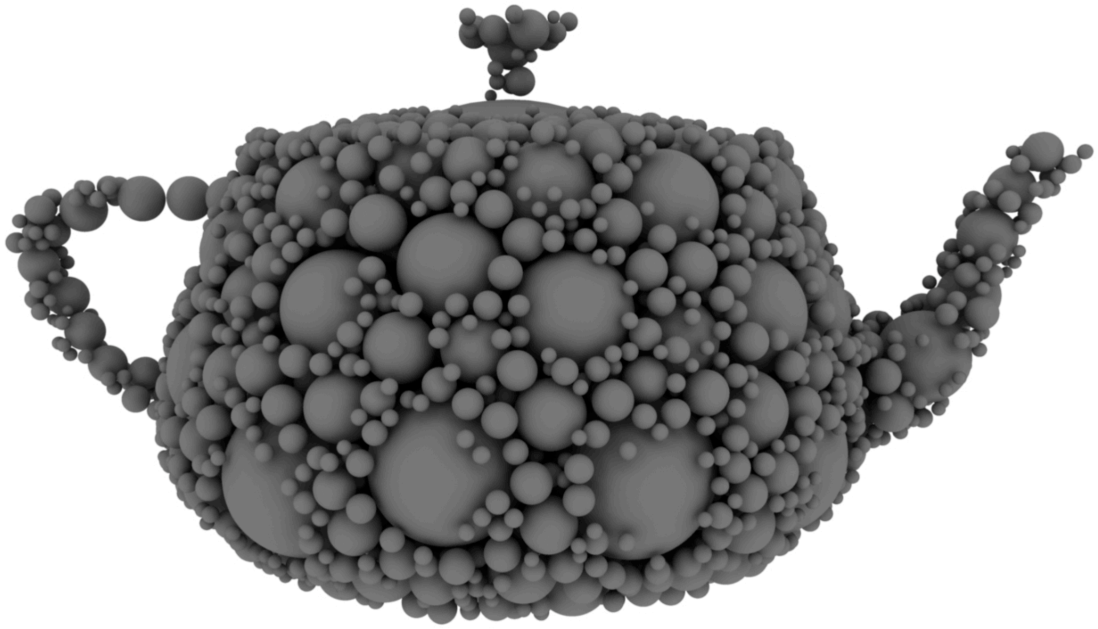


Fig. 24. Sphere elements fill 72.87% of the target volume.

CHAPTER VI

CONCLUSION AND FUTURE WORK

Our collage sculpture process has potential both as an artistic tool as well as a volume sampling algorithm. First, as we demonstrated, it can be used to make interesting 3D sculptures. However, its effectiveness as an artistic tool could be improved by a more intuitive user interface. Currently, the user must have a good understanding of the program to use it to generate a collage sculpture. However, minimally, the user only needs to interact with the collection of element shapes, the target shape, and the cutoff threshold. If a graphical user interface were hooked up to control these features, the tool could be used by a wider audience.

To further improve usability as an artistic tool, the program could be parallelized to run faster and give quicker user feedback. Both the *Placement* and *Updating* procedures loop through all the voxels and perform calculations on each independent of the rest of the system. The voxel space could be split up and handled by different parallel processes. These are the most computation-intensive steps, so parallelizing them would lead to a significant improvement in the run time of the program.

Fig. 25. Linked elements, future work.

To build on the collage sculpture process itself, one might address the issue of placing linked elements. Currently, our program will never be able to place separate elements that are connected like two links in a chain as illustrated in Fig. 25. We can place elements inside each other like a small teacup in a larger one. However with chain links, even though the interior portions of the two pieces would not overlap, the algorithm requires a contiguous sphere of open space in which to place an element, so it would not allow such a placement. This might be achieved by using more sophisticated shape descriptors for matching elements to holes.

Another limitation of the bounding sphere shape descriptors is that they do not take into account the main orientation of elements when matching them to a hole. For example, consider the head of the model in Fig. 3a. A long thin eggplant shape has been oriented to match the desired head shape. However, if our algorithm were given those elements and that target shape, the vegetables in the head would be much smaller–their length could be no more than the width of the head since their bounding sphere would have to fit entirely in the target space. Using ellipsoids instead of spheres could address this issue. An ellipsoid is directionally dependent and would require more consideration for placing in the target space. They, however, would allow for placing elements like the long thin eggplant in a matching hole. To incorporate ellipsoidal shape descriptors into our current framework, we would need to store more information in each voxel. For each voxel to store the size of ellipsoid that could be centered there, it would need to store the length and width of possible ellipsoidal holes in each direction from it. Widths of the ellipsoids at each voxel could still be calculated from the spherical holes determined by the signed distance function values. However, to calculate lengths, the updating step may need to traverse neighboring voxels to update length information.

Beyond the direct application of generating collage sculptures, our *Element*

Placement algorithm is a volume sampling method. It nicely distributes the element center points throughout the target volume. We can imagine this being useful for a number of possible computer graphics applications.

First, because the program distributes elements like the cereal pieces in Fig. 19 fairly uniformly, it could be used to place elements in an initial state for physically-based simulations. The elements are non-overlapping, so the system has a nice starting point for performing a simulation with gravity and collisions that might pack the elements together or animate them in a different way.

Bowers et al. points out a number of uses for surface sampling algorithms that exhibit a uniform and random distribution [4]. Since our algorithm distributes elements throughout the target volume so they appear randomly placed, we suggest our volume sampling method may be useful for these applications as well. These include volume texturing, remeshing volumetric data, subsurface scattering, global illumination, non-photorealistic rendering, and point-based rendering.

In conclusion, we have developed a process for generating collage sculptures by procedural rules. We implemented our process with a C++ program that assebles user-defined element into a user-defined target shape. The elements (1) visually resemble the target shape, (2) do not overlap, (3) are not deformed from their original shape, and (4) display variety in size, orientation, and position.

REFERENCES

[1] E. Akleman, “George Bush caricatures,” http://www.viz.tamu.edu/faculty/ergun/, 2011.

[2] G. Arcimboldo, “Vertumnus,” Skokloster Slott, Balsta, Sweden, 1590.

[3] P. Barla, S. Breslav, J. Thollot, F. X. Sillion, and L. Markosian, “Stroke pattern analysis and synthesis.” *Comput. Graph. Forum*, vol. 25, no. 3, pp. 663–671, 2006. [Online]. Available: http://dblp.uni-trier.de/db/journals/cgf/cgf25.html#BarlaBTSM06

[4] J. Bowers, R. Wang, L.-Y. Wei, and D. Maletz, “Parallel Poisson disk sampling with spectrum analysis on surfaces,” in *ACM SIGGRAPH Asia 2010 Papers*, ser. SIGGRAPH ASIA ’10. New York, NY, USA: ACM, 2010, pp. 166:1–166:10. [Online]. Available: http://doi.acm.org/10.1145/1866158.1866188

[5] Charlex, “Shapeshifter,” http://www.fxguide.com/featured/shapeshifting-with-charlex/, 2010, rendered frame from short film by Charlex Studio.

[6] J. H. Conway and N. J. A. Sloane, *Sphere-packings, lattices, and groups.* New York: Springer-Verlag, Inc., 1999.

[7] J. Dubuffet, “Butterfly-wing figure,” Hirshhorn Museum, Washington DC, 1953.

[8] P. Falchetta, *The Arcimboldo effect: transformations of the face from the 16th to the 20th century.* New York: Abbeville Press, 1987, ch. Anthology of Twentieth-Century Texts, pp. 207–234.

[9] A. Finkelstein and M. Range, “Image mosaics,” in *Artistic imaging and digital typography*, ser. Lecture Notes in Computer Science, R. D. Hersch, J. Andr, and H. Brown, Eds., vol. 1375. Heidelberg: Springer-Verlag, 1998.

[10] K. W. Fleischer, D. H. Laidlaw, B. L. Currin, and A. H. Barr, “Cellular texture generation,” in *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH ’95. New York, NY, USA: ACM, 1995, pp. 239–248. [Online]. Available: http://doi.acm.org/10.1145/218380.218447

[11] T. Funkhouser, M. Kazhdan, P. Shilane, P. Min, W. Kiefer, A. Tal, S. Rusinkiewicz, and D. Dobkin, “Modeling by example,” *ACM Transactions on Graphics (Proc. SIGGRAPH)*, Aug. 2004.

[12] R. Gal, O. Sorkine, T. Popa, A. Sheffer, and D. Cohen-Or, “3d collage: Expressive non-realistic modeling,” in *Proceedings of NPAR’2007*, 2007, pp. 7–14.

[13] R. Gal and D. Cohen-Or, “Salient geometric features for partial shape matching and similarity,” *ACM Trans. Graph.*, vol. 25, pp. 130–150, January 2006. [Online]. Available: http://doi.acm.org/10.1145/1122501.1122507

[14] G. Hart, “No picnic,” http://www.georgehart.com/sculpture/no-picnic.html, 1997.

[15] A. Hausner, “Simulating decorative mosaics,” in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH ’01. New York, NY, USA: ACM, 2001, pp. 573–580. [Online]. Available: http://doi.acm.org.lib-ezproxy.tamu.edu:2048/10.1145/383259.383327

[16] S.-W. Hsu and J. Keyser, "Piles of objects," *ACM Trans. Graph.*, vol. 29, pp. 155:1–155:6, December 2010.

[17] D. Huber and M. Hebert, "Fully automatic registration of multiple 3d data sets," *Image and Vision Computing*, vol. 21, no. 1, pp. 637–650, July 2003.

[18] R. Jagnow, J. Dorsey, and H. Rushmeier, "Stereological techniques for solid textures," *ACM Trans. Graph.*, vol. 23, pp. 329–335, August 2004. [Online]. Available: http://doi.acm.org/10.1145/1015706.1015724

[19] H. Jansch, "Apollo," http://www.heatherjansch.com, 2005.

[20] C. S. Kaplan and D. H. Salesin, "Escherization," in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 499–510. [Online]. Available: http://dx.doi.org/10.1145/344779.345022

[21] J. Kim and F. Pellacini, "Jigsaw image mosaics," *ACM Trans. Graph.*, vol. 21, pp. 657–664, July 2002. [Online]. Available: http://doi.acm.org/10.1145/566654.566633

[22] E. Landreneau and S. Schaefer, "Scales and scale-like structures," *Comput. Graph. Forum*, vol. 29, no. 5, pp. 1653–1660, 2010.

[23] X. Li and I. Guskov, "Multi-scale features for approximate alignment of point-based surfaces," in *Proceedings of the Third Eurographics Symposium on Geometry Processing.* Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1281920.1281955

[24] C. Ma, L.-Y. Wei, and X. Tong, "Discrete element textures," in *ACM SIGGRAPH 2011 Papers*, ser. SIGGRAPH '11. New York, NY, USA: ACM, 2011, pp. 62:1–62:10. [Online]. Available: http://doi.acm.org/10.1145/1964921.1964957

[25] T. Nowak, "Salad," http://www.framebox.de/creations/3d/salad/, 2006.

[26] A. Peytavie, E. Galin, S. Merillou, and J. Grosjean, "Procedural Generation of Rock Piles Using Aperiodic Tiling," *Computer Graphics Forum (Proceedings of Pacific Graphics)*, vol. 28, no. 7, pp. 1801–1810, 2009.

[27] P. Picasso, "Baboon and young," The Museum of Modern Art, New York, 1951.

[28] H. Piven, "Barbra Streisand," http://www.pivenworld.com/barbra-streisand/illustration, 1994.

[29] W. C. Seitz, *The art of assemblage.* New York, NY, USA: The Museum of Modern Art, 1961.

[30] L. Shapira, A. Shamir, and D. Cohen-Or, "Consistent mesh partitioning and skeletonisation using the shape diameter function," *Vis. Comput.*, vol. 24, pp. 249–259, March 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1356722.1356724

[31] P. Shilane, P. Min, M. Kazhdan, and T. Funkhouser, "The Princeton shape benchmark," in *Proceedings of the Shape Modeling International 2004*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 167–178. [Online]. Available: http://dl.acm.org/citation.cfm?id=998687.1007045

[32] N. Sloane, "The packing of spheres," *Scientific American*, vol. 250, no. 1, pp. 116–125, January 1984.

VITA

Elizabeth Grace Nemmert Muhm
c/o Ergun Akleman
College of Architecture
Texas A&M University
C108 Langford Center
3137 TAMU
College Station, Texas 77843-3137
egmuhm@gmail.com

Education

M.S., Visualization,	Texas A&M University, December 2011
B.S., Computer Science & Mathematics,	University of Washington, June 2009