# Model Creation by Velocity Controlled Surface Deformation

Risto Rangel-Kuoppa[1,2] and David Mould[1,⋆]

[1] University of Saskatchewan, Saskatoon SK S7N 5C9, Canada
rir785@mail.usask.ca, mould@cs.usask.ca
[2] Universidad Autónoma Metropolitana - Azc., Av. San Pablo No. 180,
C.P. 02200, México, D.F., México
rrk@correo.azc.uam.mx

**Abstract.** We present a scheme for the semiautomatic creation of 3D models through velocity-controlled surface deformations. Our surface representation consists of oriented points with volatile inter-neighboring point linkage. The surface is resampled in order to maintain an even distribution of points. Points created during resampling inherit their characteristics from their progenitors. Our surface representation and resampling behavior support detailed irregular surfaces with smooth transitions such as those of organic entities. Surface velocities are set by the combination of two types of operators, selection and velocity assignment, with operator application managed by a finite state machine. We demonstrate our scheme with the creation of some branched, fruit-like and mushroom-like models.

## 1 Introduction

We present a velocity-based surface deformation control scheme for model creation. Modeling is achieved by beginning with a simple surface such as a sphere and progressively deforming it to add shape and detail. Our surfaces are represented by a collection of oriented points; each point has a link to every other nearby point. The surface deformation is specified by a set of velocity operators applied to its points, and these operators are in turn managed by a velocity control scheme. Our surface deformation system includes automatic spatial density resampling; surface resampling creates new surface elements that inherit their attributes from the surface elements which spawned them. Our scheme is suited to creating models consisting of detailed surfaces with smoothly varying irregularities, especially organic models such as plants. Figure 1 depicts some of the above concepts and shows a model created by our method.

⋆ We thank and acknowledge the *Universidad Autónoma Metropolitana U. Azc.-D. CBI* and *CONACyT (Mexican Council for Science and Technology)* for sponsoring Risto Rangel-Kuoppa PhD studies. This work was supported in part by NSERC RGPIN299070-04.

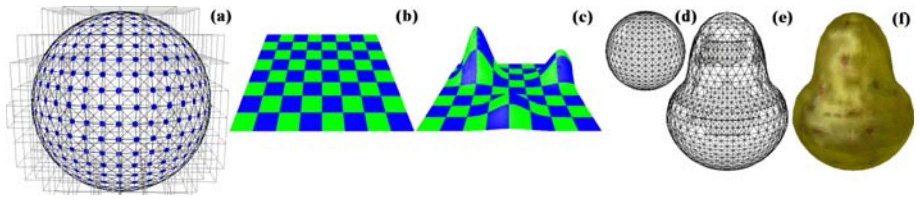

**Fig. 1.** (a) A spherical point cloud rendering with its inter-point neighboring linkage and space partitioning. (b,c) Flat surface deformation to demonstrate surface elements' feature inheritance and normal computation during resampling. (d) An initial sphere. (e) Sphere deformed to a pear shape. (f) Resulting model with pear texture.

## 1.1 Previous Work

Surface deformation has been previously investigated for both polygonal mesh and point-based surface representations. One approach for modifying polygonal meshes is to deform the mesh components: edges and vertices. For example, Combaz and Neyret [1] "paint" stressing forces onto the surface, leading to the elongation or contraction of the edges, and generating folds. Similarly, Lawrence and Funkhouser [2] "paint" velocities on the surface elements and simulate the resulting movement to obtain various models. Both their approaches and ours use the concept of applying velocities to surface elements. We avoid the self-intersection problem, present in both Combaz and Neyret's work and Lawrence and Funkhouser's, by using spatial subdivision to detect self-intersections and correct them by fusing samples.

Techniques for rendering point-based surfaces were reported by Pfister et al. [6] and Rusinkiewicz and Levoy [7] who used oriented points and a space partitioning structure to control the level of detail. This type of surface has also been used as a platform for model creation. Pauly et al. [5], Zwicker et al. [10], and Szeliski and Tonnesen [9] used oriented particles to define their surfaces and provided deformation mechanisms where local or inter-point information was used for resampling purposes. Our approach builds on some of the concepts provided by the previous authors. We determine our surface deformation based on a collection of velocity operators, rather than requiring a user to paint velocity or (in the case of PointShop3D [10]) to manipulate the points interactively. Unlike Zwicker et al., but like Lawrence and Funkhouser, we allow points to have velocities in arbitrary directions. Unlike pure point-based methods, we include in our surface model neighboring inter-point connectivity information that is used to generate the final surface model and to inherit point characteristics during resampling. The surface model of Szeliski and Tonnesen [9] considers the resampling of the surface when its components are too far away from each other, but their surface elements suffer from additional displacements because of the inter-point attracting and repelling forces they used. This situation does not occur in our framework since the inter-point connectivity information is used only to resample the surface and not to alter the positions of surface elements.

## 2 Algorithm

We define our surface as a set of oriented points. Each point has a linkage to its neighbors; two points $p_i$ and $p_j$ are neighbors if they are closer than $\rho$ units. The neighboring linkage is later used for surface resampling. Note that the linkage does not define a polygonal mesh because a triangulation of the points and their linkage would include intersecting and overlapping triangles; nevertheless, a manifold surface would be a subset of this triangulation. Our surface definition lies between the pure point surface definitions [5, 6, 7, 9, 10] and the more traditional polygonal mesh representations.

We accomplish surface deformation by associating velocities with individual points of the surface. These velocities are used to update the points' positions at each simulation step (see Fig. 2) and since the velocities may produce stretching and compressing displacements, resampling is required. Dynamic resampling occurs by reacting to the distance between registered neighbors: if the neighbors are farther than $R_{\text{Max}}$ or closer than $R_{\text{Min}}$ units then a *splitting* or *fusion* operation is respectively triggered.

The resampling operations generate new intermediate points between the pair of points whose distance value triggered the operation. The position of the new point is obtained by linearly interpolating between the old points. Note that our resampling scheme is different from a polygonal mesh resampling [4] since the relations between points may not correspond to a triangulated mesh. Our resampling is also different from those of point-only surface representations [5, 8, 10] because we do not approximate the new points from a subset of points; rather, we only use the spawning points' information.

The points created due to a resampling operation inherit all other characteristics from their parents (including position and velocity), but the normal (orientation) requires a different computation, described as follows and depicted in Fig. 3. First, we create a plane $P_l$ whose normal is the difference between the spawning points positions, $p_{\text{diff}} = \text{norm}(p_j - p_i)$. Next, we take an approximate normal $N_{\text{le}}$ by linearly interpolating the spawning points' normals. Finally, the

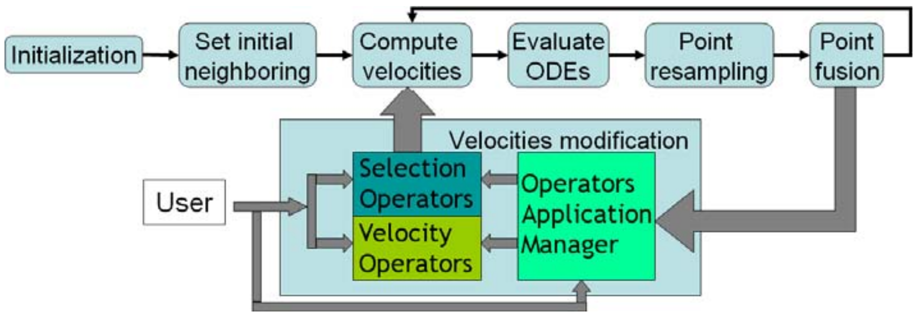

**Fig. 2.** Surface deformation simulation cycle; note that the individual point velocity evaluations and surface resampling are executed in parallel

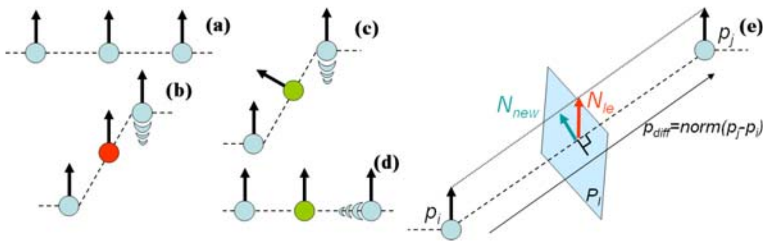

**Fig. 3.** (a) Initial surface, (b) Wrong normal Computation, (c) and (d) proper normal, (e) How normal is computed

normal of the new point is the normalized projection of $N_{\mathrm{le}}$ on the plane $P_{\mathrm{l}}$, i.e., $N_{new} = \mathrm{norm}(N_{\mathrm{le}} - c \cdot p_{\mathrm{diff}})$ where $c = N_{\mathrm{le}} \cdot p_{\mathrm{diff}}$, as shown in Fig. 3.

We will next discuss assigning velocities to the points. We define two types of operators: Selection Operators (SO) and Velocity Operators (VO); the SO selects a group of points and the VO changes the velocities of points to which it is applied. Both types of operators are specified in terms of any measurable geometric property of the surface or its points, as well as accepting numerical parameters; for example, "All points reachable by traversing $n$ links from point $p_{\mathrm{k}}$" would be an SO, and "A point's velocity is twice its distance to the surface centroid" a VO. Typically we apply a VO to the points selected by an SO.

We have used two operators in the examples in this paper. The first one is the "Scaled Unitary Gaussian" operator represented by $SUG(p_{\mathrm{k}}, \sigma, d)$ where $p_{\mathrm{k}}$ is a surface point, $\sigma$ a standard deviation value, and $d$ a distance. The SUG is a two stage operator. The first stage is an SO that chooses all points whose Euclidean distance is less than $d$ units from $p_{\mathrm{k}}$. The second stage modifies the chosen points' velocities by adding a velocity in the normal direction, scaled by a normalized Gaussian distribution centered in $p_{\mathrm{k}}$, i.e., $v_{p_i} = v_{p_i} + n_{p_i} \cdot e^{-\frac{\mathrm{dist}(p_k, p_i)^2}{2 \cdot \sigma^2}}$. The overall effect of the SUG operator is to increase the velocity of points surrounding $p_k$ along their normal direction by a magnitude that falls off as $e^{-\mathrm{x}^2}$.

The second operator is the Voronoi Regionalization $VR(m, t)$ operator, which is an SO only. Using a variant of Lloyd's algorithm [3], this operator creates $m$ subsets of points making $t$ passes of the algorithm; the subsets correspond to centroidal Voronoi regions. Applying this operator to a model results in a partition of the model, which can later be used by another operator. For example, in Fig. 4.a we can see the results of applying $VR(13, 1)$ and in Fig. 4.b we see $VR(13, 10)$. In Fig. 4.c we have used the centers of the regions created with $VR(13, 10)$, say $p_1, \ldots, p_{13}$, and applied $SUG(p_i, 0.9, 8.0)$ $i = 1, \ldots, 13$. In Fig. 4.c the points color intensities are proportional to their velocities. Finally, Fig. 4.d shows the surface after it has evolved into a surface with Voronoi-distributed bumps; each bump has a smooth transition to the flat surface.

Even though just having a library of operators would be very useful for the creation of different models, a user would still need to apply them individually. As part of our scheme we provide an Operator Application Manager (OAM) to control which operators are applied and when and where they are applied.

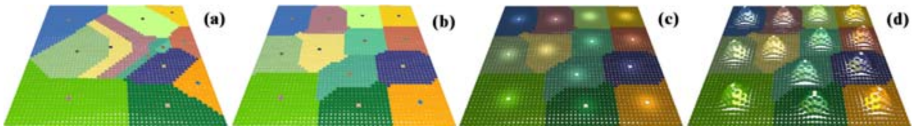

**Fig. 4.** Simple deformation operators used in series to achieve a complex deformation

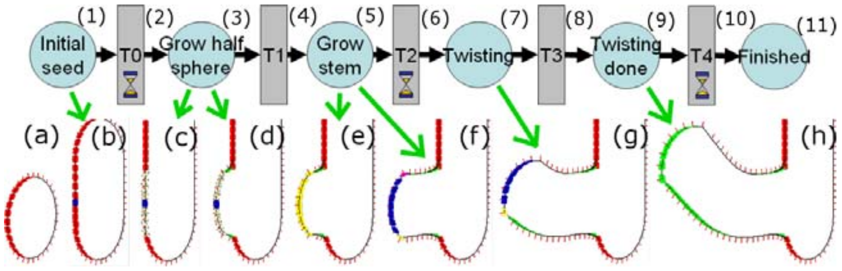

**Fig. 5.** Above, an OAM net controlling the operators to be applied. Below, application of the OAM net

The OAM controls the operators through a finite state machine. An OAM is similar to a Petri Net, the difference being that the transition elements in the OAM can be fired when only one of the input states contains a token. The OAM net transitions are constantly monitoring a specified point cloud metric; possible metrics include maximum height, number of subregions, or time elapsed. The OAM net states can be associated with a VO. Figure 5 shows an OAM net that is used to grow a branch that later twists to a given angle. With the proper application of the operators and the OAM nets, our system can simulate a variety of effects. Global effects like gravity, local effects such as tropism, or local modifications to the mesh such as branches can all be described within our system.

We next describe the behavior of the OAM net in Fig. 5. This net contains only one token, initially placed in the state *Initial seed*.

1. State *Initial seed* – Applies the SO CreateRadialSubset$(P_i, r)$ which selects the reachable points in a radius $r$ from $P_i$ and records the current position of $P_i$, $\overrightarrow{p_0} = \overrightarrow{p_i}$.
2. Transition *T0* – Timed transition firing after 0 seconds.
3. State *Grow half sphere* – Applies a VO that assigns to each of the points selected in the previous state a normalized velocity proportional to the difference of their distance to $P_i$ and $r$, i.e., $\overrightarrow{V_{\text{new}}} = \overrightarrow{N} \cdot (r - \|\overrightarrow{p_x} - \overrightarrow{p_i}\|)$
4. Transition *T1* – Fires when the distances from all selected points to the original position of $P_i$, $\overrightarrow{p_0}$, are greater than $r$.
5. State *Grow stem* – Applies a VO setting the selected points' velocities to have the same direction as their normals.

6. Transition *T2* – Use an VO to make the selected points' velocities equal to zero and compute a pivoting place in space $\overrightarrow{T} = \overrightarrow{p_x} - (Y_{\text{offset}} \cdot \widehat{j})$ where $Y_{\text{offset}} \geq r$.
7. State *Twisting* – Rotates the selected points upward using pivot $\overrightarrow{T}$.
8. Transition *T3* – Fires when all the selected points have rotated $\gamma$ degrees.
9. State *Twisting done* – Applies the same operator used in state *Grow stem.*
10. Transition *T4* – Fires after $t4$ seconds
11. State *Finished* – Marks the end.

All of the concepts described in this section have been implemented in a software tool. This tool has an interface similar to that of Lawrence and Funkhouser [2] with which a user can specify the parameters of an initial surface. The surface can be deformed by either applying velocities to individual points selected by the user or applying the deformation operators. The deformation is controlled by the user and can be paused at any time; also the user can apply new velocities or operators at any moment. This software tool was used to create the models shown in the following section.

## 3 Results

Using the surface specification, operators, and OAM defined previously, we have created a set of models. Figure 6.a-f depicts the steps involved in the creation of an apple model. First, following a ring path, an SUG operator is applied to the upper half of the initial sphere. Next, four instances of the same operator are applied on the lower half plus a negative weighted SUG is applied to the lowermost point, creating a crown-like section. Figure 6.c shows the surface evolved in time. Two additional Gaussian-like velocity distributions are applied to the upper half to mimic the heart-shaped contour that some apples present (Fig. 6.d). Figure 6.e-f shows the final point cloud and its connectivity mesh. More complex models can be obtained if the operators are applied to subsets of the point cloud. Figure 6.g-l shows the steps to create the basic structure of mushroom-like models.

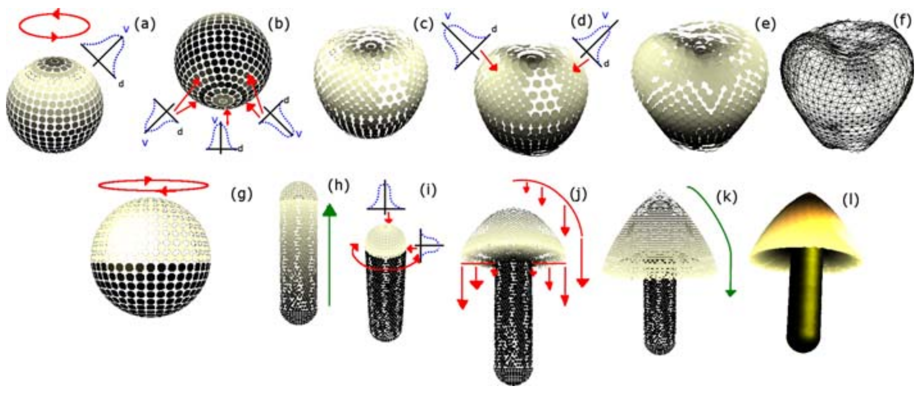

**Fig. 6.** (a-f) Apple-like model creation, (g-l) Mushroom-like models structure creation

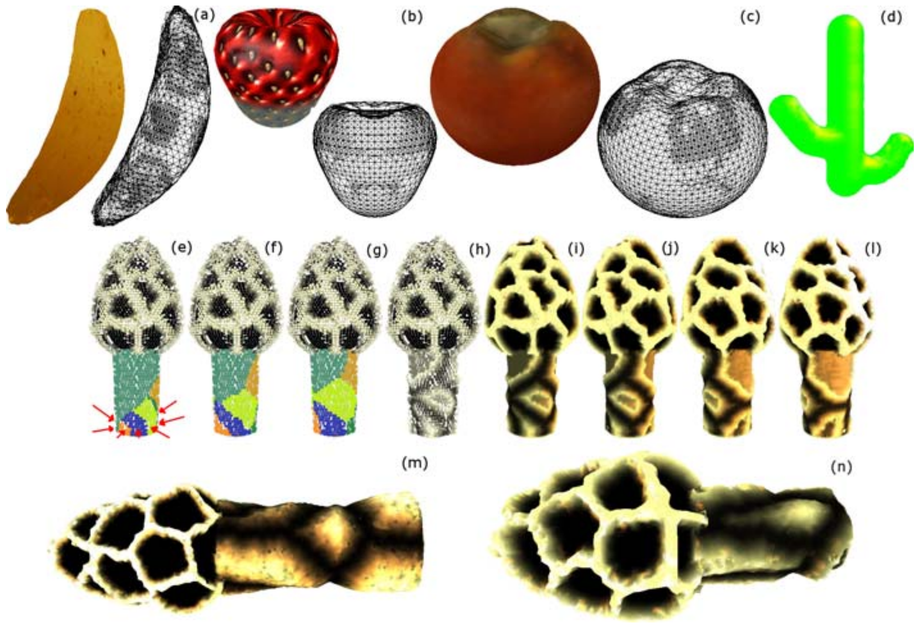

**Fig. 7.** (a-d) Fruit-like models, (e-h) Mushroom model detail creation, (i-l) Rotated views of mushroom model, (m-o) Close ups of mushroom model

Additional examples created with similar techniques are shown in Fig. 7.a-c. Figure 7.d was generated with two of the OAM nets shown Fig. 5. One of our favorite models is shown in Fig. 7.e-n. Initially we have a mushroom model with a stem and an egg-shaped cap to which a Voronoi regionalization operator is applied. Figure 7.e-g shows the result of the Voronoi regionalization operator with initial seeds randomly placed on the stem's base. In Fig. 7.h, the points' velocity is set as a function of their distance to the center of their respective Voronoi region; if the velocity is greater than a certain threshold, then the velocity to apply is zero. Finally, in Fig. 7.m and Fig. 7.n we can see two different models created by varying the parameters previously described as well as a different initial mushroom-like structure. The fruit models contain around 1500 points and the mushrooms have about 27000. Simulation times for the fruits were approximately 5 seconds, and for the mushrooms of Fig. 7.e-n, approximately 30 minutes. All models were created on a 3 GHz Pentium 4 computer with 1 GB RAM.

## 4 Conclusions and Future work

We have presented a novel model for surface deformation and its use with a semi-automatic model generation scheme. The surface specification can be considered a hybrid between point based surfaces and triangulated meshes because the surface is represented by a point cloud with connections between points. We do not have to maintain consistency as with a mesh, but we do have neighborhood information for performing local operations on the surface, unlike a pure point cloud. We have defined some basic operators and provided specific detail on two specialized operators: application of a Gaussian velocity distribution and Voronoi regionalization of point subsets. We showed some models whose surfaces have irregularities with smooth transitions, the type of surfaces commonly present in natural models. As part of our continuing research we are looking to implement different operators that reflect other natural phenomena of growing entities, such as bark and lichen growth, and we are also aiming to simulate aging effects on surfaces like cracking and erosion.

## References

1. Combaz J. and Neyret F., *Painting Folds using Expansion Textures*, Pacific Graphics, 2002, vol. 1, no. 1, pp. 176-183, October
2. Lawrence J. and Funkhouser T., *A Painting Interface for Interactive Surface Deformations*, 11th Pacific Conference on Computer Graphics and Applications (PG'03), 2003, vol. 1, no. 1
3. Lloyd S., *Least squares quantization in PCM*, IEEE Transactions on Information Theory, 1982, vol. 28, no. 2, pp. 129-137
4. Mandal Chandomay, Qin H. and Vemuri B., *Dynamic modeling of butterfly subdivision surfaces*, IEEE Transactions on Visualization and Computer Graphics, 2000, vol. 6, no. 3, pp. 265-287
5. Pauly M., Keiser R., Kobbelt L.P. and Gross M., *Shape modeling with point-samples geometry*, Proceedings of the ACM SIGGRAPH 2003, 2003, vol. 22, no. 3, pp. 641-650, July
6. Pfister H., Zwicker M., van Baar J. and Gross M., *Surfels: Surface elements as rendering primitives*, Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques SIGGRAPH'00, 2000, vol. 1, no. 1, pp. 335-342
7. Rusinkiewicz S. and Levoy M., *QSplat: A multiresolution point rendering system for large meshes*, Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH'00, 2000, vol. 1, no. 1, pp. 343-352
8. Sethian J., *Level Set Methods*, United States of America: Cambridge University Press
9. Szeliski R. and Tonnesen D., *Surface Modeling with Oriented Particle Systems*, Computer Graphics, 1992, vol. 2, no. 26, pp. 185-194, July
10. Zwicker M., Pauly M., Knoll O. and Gross M., *Pointshop3D: An interactive system for point-based surface editing*, Proceedings of the 29th Annual Conference on Computer Graphics and INteractive Techniques (SIGGRAPH'02), 2002, vol. 1, no.1, pp. 322-329