Journal of Zhejiang University SCIENCE A
ISSN 1673-565X (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Interactive point cloud blending by drag-and-drop[*]

ZOU Wan-hong[†1], DING Zhan[1], YE Xiu-zi[†1], CHEN Zhi-yang[2]
([1]*State Key Lab. of CAD and CG, School of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China*)
([2]*Software College, Zhejiang University of Technology, Hangzhou 310014, China*)
[†]E-mail: wh_zou@zju.edu.cn; yxz@zju.edu.cn

Received Jan. 10, 2007; revision accepted May 15, 2007

Abstract: With the rapid development of 3D digital photography and 3D digital scanning devices, massive amount of point samples can be generated in acquisition of complex, real-world objects, and thus create an urgent need for advanced point-based processing and editing. In this paper, we present an interactive method for blending point-based geometries by dragging-and-dropping one point-based model onto another model's surface metaphor. We first calculate a blending region based on the polygon of interest when the user drags-and-drops the model. Radial basis function is used to construct an implicit surface which smoothly interpolates with the transition regions. Continuing the drag-and-drop operation will make the system recalculate the blending regions and reconstruct the transition regions. The drag-and-drop operation can be compound in a constructive solid geometry (CSG) manner to interactively construct a complex point-based model from multiple simple ones. Experimental results showed that our method generates good quality transition regions between two raw point clouds and can effectively reduce the rate of overlapping during the blending.

Key words: Drag-and-drop, Point cloud model, Interactive blending, Geometry modeling
doi:10.1631/jzus.2007.A1633 **Document code:** A **CLC number:** TP391.72

INTRODUCTION

In recent years, point primitives have received growing attention in computer graphics. There are two main reasons for this new interest in points. On one hand, we have witnessed a dramatic increase in the polygonal complexity of computer graphics models. The overhead of managing, processing and manipulating very large polygonal meshes has led many researchers to question the future utility of polygons as the fundamental graphics primitive. On the other hand, modern 3D digital photography and 3D scanning systems facilitate the ready acquisition of complex, real-world objects. These techniques generate huge volumes of point samples and have created the need for advanced point processing.

In an alternative pipeline for efficient 3D content creation, after a surface model has been reconstructed (a powerful mathematical representation that interpolates the discrete point samples), users will want to edit the surface geometry or modify its appearance attributes. Point-based editing can be classified as either editing appearance of point-based 3D model or modeling geometry information.

The open-source point-based surface editing system Pointshop 3D (Zwicker *et al*., 2002) implements re-texturing, sculpting, embossing and filtering, in the same way as we know from conventional 2D image editing. Overall, the system combines the efficiency of 2D photo editing with the functionality of 3D sculpting systems. Adams *et al*.(2004) presented a novel painting system for 3D objects. In contrast to mesh-based painting systems, an efficient dynamic re-sampling scheme permits arbitrary levels of painted detail.

In contrast to appearance editing, point-based geometry modeling is always related to the repre-

[*] Project supported by the National Natural Science Foundation of China (Nos. 60473106 and 60333010), and the Program for Changjiang Scholars and Innovative Research Team in University (No. IRT0652), China

sentation of point-based surface. Combining with the particle sampling technique, Radial basis function (RBF) based implicit surface representation of point sampled surface was presented by Turk and O'Brien (2002). They described some basic low-level operations such as moving a point, adding a point, or changing a normal, which can be used to get free-form modeling, with high-level operations such as Boolean operation and blending also being introduced. Because their point-sampled surface representation is not efficient, Turk and O'Brien (2002)'s modeling system is slow and cannot provide interactive frame rates after surface changes. Based on a new point-based rendering algorithm, Reuter *et al.*(2003) implemented a system rendering in a view-dependent manner without the creation of a polygonal mesh representation for implicit surface. This enhancement eventually enables interactive modeling of the entire surface. More recent work on RBF based implicit surface geometry modeling was introduced by Botsch and Kobbelt (2005), who proposed to use tri-harmonic RBFs for real-time freeform shape editing. An incremental least-square method enables approximate solution of the linear systems involved in a robust and efficient manner, and by pre-computing a special set of deformation basis functions we can significantly reduce the per-frame costs. Moreover, evaluating these linear basis functions on the GPU finally allows them to deform highly complex polygon meshes or point-based models at a rate of 30 M vertices or 13 M splats per second, respectively.

The moving least-squares (MLS) surfaces of (Levin, 2003) provide an approximating or interpolating surface for a given set of point samples by local higher order polynomials and was first applied to point-based methods by Alexa *et al.*(2001; 2003). Based on a new representation of point-based surface by combining unstructured point clouds with MLS surface, Pauly *et al.*(2003) introduced a shape modeling system that enables the designer to perform large constrained deformations as well as Boolean operations on arbitrarily shaped objects.

Found upon the rigorous mathematics of Riemann surface theory and Hodge theory, Guo *et al.*(2006) presented global parameterization for point-based geometry. Within their parameterization framework, any well-sampled point surface is functionally equivalent to a manifold, enabling popular and powerful surface-based modeling and physically-based simulation tools to be readily adapted for point geometry processing and animation.

Geometry modeling independent of point-based surface representation was presented in (Adams and Dutre, 2003a), where an algorithm to perform interactive Boolean operations on free-form solids bounded by surfels is proposed. After constructing an adaptive three-color octrees, Yang *et al.*(2005) presented a more robust Boolean operation for general point-based geometry including noisy point models, non-uniform sampled and different sampling resolution point models.

Adams and Dutre (2003b) proposed a method for blending two point set surface by a smooth operation. Their smooth operation is simple but it works as a post process of the complicated Boolean operation and needs much information from the Boolean operation. Pauly *et al.*(2003) also implemented an adaptation of oriented particles to smooth out the sharp creases created by Boolean operations. But their method is hard to implement. Although blending or fusion is always used to smooth the objects after Boolean operation in point-based geometry, it can be considered as new modeling tools by "cutting" and "pasting" several models into one model in mesh-based geometry (Liu *et al.*, 2005; Jin *et al.*, 2006). Jin *et al.*(2006) presented a new method for mesh fusion. After converting the sections with boundaries of the under-fusing meshes into implicit representations, an implicit transition surface is created based on cubic Hermit blending to join the sections together while maintaining the smoothness along the boundaries. Finally, the implicit surface is tessellated to form a resultant mesh. Besides mesh fusion, Fang *et al.*(2000) proposed volume fusion and volume graphics, the voxelization algorithms can be integrated into a more systematic volume fusion system that provides a uniform framework for the interactive modeling and rendering of volumetric scenes.

In this paper, we present an interactive method for blending point-based geometries by dragging-and-dropping one point-based model onto another model's surface metaphor. RBF is used to construct an implicit surface to smoothly interpolate the transition regions. The drag-and-drop operation can be compound in a CSG (constructive solid geometry) manner to interactively construct a complex point-based model from multiple simple ones. Experimental results showed that our method generates good quality transition regions between two raw point clouds and can effectively reduce the rate of the overlapping during the blending. We would like to emphasize that our method takes raw point cloud as the input, with only points' coordinate being available, and we only calculate points' normal when it is required.

The paper is organized as follows. Section 2 introduces the pipeline of our interactive point-based blending system. Section 3 presents the algorithm for dragging one point-based model on another model's surface metaphor, which lies on the core of our interactive blending system. Section 4 describes how to calculate the regions for blending, and Section 5 gives the RBF based construction of transition regions to smoothly interpolate the blending regions. In contrast to mesh blending, meshes and the transition regions will be combined into one model. Section 6 gives some experimental results, and Section 7 draws conclusions and presents our future work.

INTERACTIVE BLENDING PIPELINE

We first introduce some abbreviations that will be used throughout the paper.

M_p represents a point model, which is the primary entity for blending, and also the referring model adjusting 3D position. M_c denotes a child point model, which is the point model to be dragged and blended together with M_p. P_{inter} denotes the polygon of interest, which is a polygon depicted by the user to express the blending scope. T_R describes a transition region which is a smooth implicit surface that interpolates the blending regions.

As shown in Fig.1, our interactive blending scheme can be described as follows:

(1) "Cutting" point model M_c from some other point model (Fig.1a).

(2) Finding the polygon of interest P_{inter} on M_c, then "Pasting" M_c to surface of M_p (Fig.1b).

(3) Dragging M_c freely on surface of M_p to some position P, calculating blending region F_c in M_c and F_p in M_p, then constructing transition region T_R (Fig.1c).

(4) If needed, adjust the 4 degrees of freedom (DOFs) of M_c at P, recalculating F_c and F_p to construct a new transition region T_R. Repeating (3) and (4) (Fig.1d).

(5) Combining T_R, M_c and M_p into one model M_b (Fig.1e).

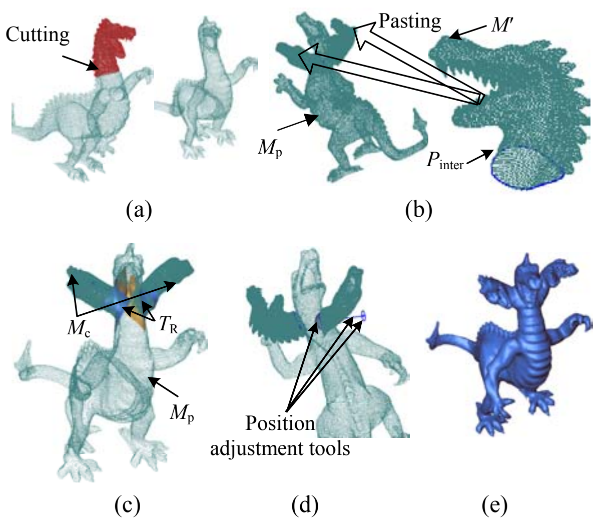


Fig.1 Schematics of interactive point model blending modeling. (a) Cutting; (b) Pasting; (c) Construction of transition region; (d) Position adjustment; (e) Combining transition region with the orginial models

INTERACTIVE MODELING TOOLS

Freely adjusting model position is an essential part of multi-model blending system. Due to the difference between 2D view plane and 3D model coordinate system, it is hard for user to adjust model to the desired position. Experiences show that it is more convenient to adjust model position referred to to another model, i.e., the model to be blended with. Smooth Teddy (Igarashi and Hughes, 2003) implemented a similar interactive dragging metaphor on mesh surface. Here we present a 3D position adjusting metaphor in which the dragging model is always attached to the referred model's surface in the whole process. In addition, the user can modify the 4 DoFs attached to the movable model after it was dropped on to the other model's surface. Experiments show that with our 3D position adjustment tools, the user can quickly and freely perform the blending operation.

Drag-and-drop

Dragging an object can be considered as the process of transforming one initial position-normal pair ($\boldsymbol{P}_{\mathrm{I}},\boldsymbol{V}_{\mathrm{I}}$) into the target position-normal pair ($\boldsymbol{P}_{\mathrm{T}},\boldsymbol{V}_{\mathrm{T}}$) with translation and rotation. So the essential task is to compute ($\boldsymbol{P}_{\mathrm{I}},\boldsymbol{V}_{\mathrm{I}}$) and ($\boldsymbol{P}_{\mathrm{T}},\boldsymbol{V}_{\mathrm{T}}$). The dragging process consists of two steps, namely, pasting M_{c} to the surface of M_{p}, and continuing dragging on M_{p}. The methods for computing ($\boldsymbol{P}_{\mathrm{I}},\boldsymbol{V}_{\mathrm{I}}$) are different in the two steps.

After “cutting” M_{c} from one model, it is important for the user to figure out the blend region, namely the region of interest, on M_{c}. The boundary of region of interest is denoted by P_{inter}, which usually is the open boundary by the “cutting” of M_{c} from its original point model. Such open boundary can be detected by boundary detecting algorithm (Qian *et al*., 2005) on point model. A user can select the detected boundary or use freehand sketching tools to depict P_{inter}. After P_{inter} is obtained, $\boldsymbol{P}_{\mathrm{I}}$ can be taken as the centroid of P_{inter}, and $\boldsymbol{V}_{\mathrm{I}}$ be the normal vector of the plane that least-square fits P_{inter}. It is clear that the target position-normal pair ($\boldsymbol{P}_{\mathrm{T}},\boldsymbol{V}_{\mathrm{T}}$) in the current dragging operation becomes initial value of the next dragging operation.

Computing ($\boldsymbol{P}_{\mathrm{T}},\boldsymbol{V}_{\mathrm{T}}$) based on discrete depth buffer

In the dragging operation, after pressing left-mouse-button (LMB) down and moving cursor from a 2D position $\boldsymbol{P}_{\mathrm{p2d}}$ to a 3D point $\boldsymbol{P}_{\mathrm{p3d}}$ on the surface of M_{p}, $\boldsymbol{P}_{\mathrm{T}}$ becomes $\boldsymbol{P}_{\mathrm{p3d}}$ and $\boldsymbol{V}_{\mathrm{T}}$ is the normal of the surface at $\boldsymbol{P}_{\mathrm{p3d}}$. A hardware based discrete depth buffer algorithm is used here to efficiently identify $\boldsymbol{P}_{\mathrm{p3d}}$ and its normal from $\boldsymbol{P}_{\mathrm{p2d}}$. Depth buffer based algorithm is characterized by calculating the 3D position in the World Coordinate System (WCS) from Graphic Device Interface (GDI) point, based on the model’s current rendering parameters such as view matrix, projection matrix, view port and depth buffer. In the same manner, 3D positions will be calculated from the GDI points that are close to $\boldsymbol{P}_{\mathrm{p2d}}$, and a least-squares plane will be fitted to these 3D points.

Before calculating $\boldsymbol{P}_{\mathrm{p3d}}$ from $\boldsymbol{P}_{\mathrm{p2d}}$, we need to prepare the depth buffer first. As the user presses LMB inside boundary box of M_{c}, the algorithm starts to do the preprocessing. The system first cleans the depth buffer, and then renders M_{p}, which becomes the depth buffer for M_{p}. Fig.2b is the depth buffer for point model in Fig.2a. Fig.2g is mesh of the point model after triangulation, and Fig.2h is the corresponding depth image. It can be seen that the depth image of point model is not continuous, but the mesh’s depth image is.

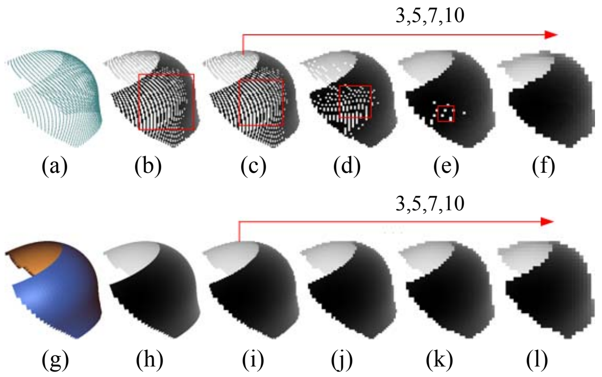


Fig.2 Raw depth images and discrete sorted depth images of point cloud and mesh. (a)~(f) are point clouds, (g)~(l) are meshes. Lighter dots mean that the points are closer to the view plane

The use of depth images for calculating $\boldsymbol{P}_{\mathrm{p3d}}$ from $\boldsymbol{P}_{\mathrm{p2d}}$ can cause the problem that two close cursor positions can result in 3D positions far away, and produces poppy effects in dragging. In Fig.3, two 2D cursor point $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ are supposed to produce two 3D points on the front surface of the point model (rectangles in the figure). Because of the discontinuity of the depth image, one 3D point is at the deepest distance, and the other is on the back surface of the point model. To overcome this defect, we discretize the depth buffer to make the depth image in a visually continuous fashion as follows:

Step 1: Partition the current depth buffer into continuous blocks, with each block size being $n\times n$, where n is a discretization resolution.

Step 2: Iterate each block, and set each block’s depth to the minimum depth of this block.

Step 3: Allocate a block of memory M to record these blocks and corresponding depth, M will be used in the operations that follow.

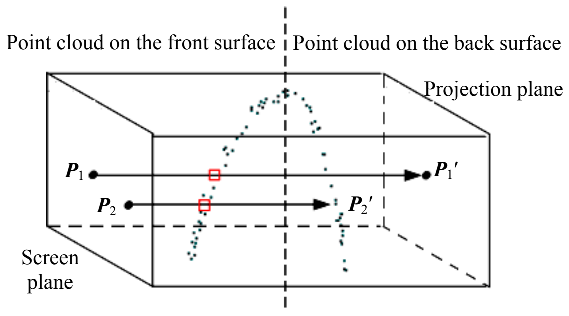

Point cloud on the front surface
Point cloud on the back surface
Projection plane
P_1
P_1'
P_2
P_2'
Screen plane

Fig.3 Incorrect reversed-projection based on depth buffer

Figs.2c~2f show different depth buffers using different resolution n on the same depth buffer in Fig.2b. As expected, when n increases, the depth buffer becomes more and more continuous; till n=10, it becomes completely continuous visually. A visually continuous depth buffer will give a smooth dragging of M_c on M_p's surface. In this paper, n is set to 10. Grossman and Dally (1998) presented a pull-push algorithm to fill gaps of depth buffer. Our discrete buffer algorithm is similar to the pull-push algorithm, but only has the pull stage and weights of different pixels are set to 1 or 0. So our algorithm is simpler and faster than the pull-push algorithm.

Our depth buffer based algorithm can sometimes go wrong. Once LMB is pressed down and starts to move, the dragging algorithm is activated. As mentioned, $\boldsymbol{V}_I$ is taken to be the centroid of the 3D coordinates of P_{inter}. Let the centers of the 2D GDI and cursor positions P_{inter} be $\boldsymbol{C}_{2d}$ and $\boldsymbol{P}_{2d}$, respectively. When the cursor is moved to a new position $\boldsymbol{P}_{n2d}$, we would expect $\boldsymbol{C}_{2d}$ to move to a new position $\boldsymbol{C}_{n2d}=\boldsymbol{C}_{2d}+(\boldsymbol{P}_{n2d}-\boldsymbol{P}_{2d})$. This may not be always true in practice since M_c is required to stay on the surface of M_p. As illustrated in Fig.4, if the cursor is out of M_p's surface, the new position becomes $\boldsymbol{C}_{n2d}'$ rather than $\boldsymbol{C}_{n2d}$.

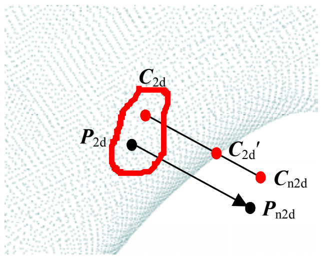


Fig.4 Centroid $\boldsymbol{C}_{n2d}$'s actual and theoretical positions

We fix the above problem by discretizing the line segment $\boldsymbol{C}_{2d}\boldsymbol{C}_{n2d}$ using Bresenham algorithm. $\boldsymbol{C}_{n2d}'$ is chosen to be the last point on surface of M_p along $\boldsymbol{C}_{2d}\boldsymbol{C}_{n2d}$, whose depth value is less than 1.0, and $\boldsymbol{P}_T$ can then be correctly calculated from $\boldsymbol{C}_{n2d}'$ and its depth value is in discrete depth buffer M.

$\boldsymbol{V}_T$, the normal vector associated with $\boldsymbol{P}_T$, can be computed by fitting a least square plane through $\boldsymbol{P}_T$ and its neighbors. With the (reasonable) assumption that $\boldsymbol{P}_T$ and its neighbors have continuous depth values after discretization of the depth buffer, $\boldsymbol{P}_T$'s neighbors can be obtained (Fig.5) as follows. Let the block index of $\boldsymbol{C}_{n2d}'$ in M be ID, the 8 connected neighbors of ID in M are searched. The depth of $\boldsymbol{C}_{n2d}'$'s surface neighboring points is less than 1.0. In Fig.5, seven points whose depths are less than 1.0 will be used in the least square fitting. After ($\boldsymbol{P}_I$,$\boldsymbol{V}_I$) and ($\boldsymbol{P}_T$,$\boldsymbol{V}_T$) are obtained, we can make the transform. We first rotate P_{inter} and M_c along the axis ($\boldsymbol{P}_I$,$\boldsymbol{V}_T\times\boldsymbol{V}_I$) with an angle $\alpha=\boldsymbol{V}_T\times\boldsymbol{V}_I$. Then we translated P_{inter} into a vector $\boldsymbol{P}_T-\boldsymbol{P}_I$ (refer to Fig.6), and then update the position-normal pair by setting the current target position-normal pair ($\boldsymbol{P}_T$,$\boldsymbol{V}_T$) to the new ($\boldsymbol{P}_I$,$\boldsymbol{V}_I$).

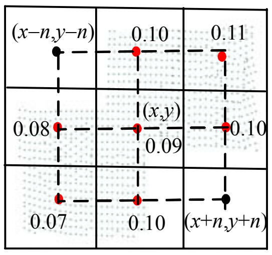


Fig.5 Neighbor searching for computing normal vector

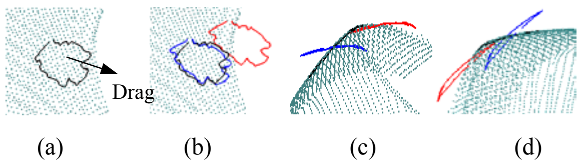


Fig.6 Transformation after drag operation. (a) Drag; (b) Translation and rotation. (c) and (d): Observing from different viewpoints

Local position adjustment

The user can adjust M_c to any point on surface of M_p with the dragging metaphor. In many cases, the user still needs to adjust M_c locally, e.g., rotating M_c around the centroid or translating along the normal vector of centroid. We provide this joystick-like local position adjustment method as follows (Fig.7):

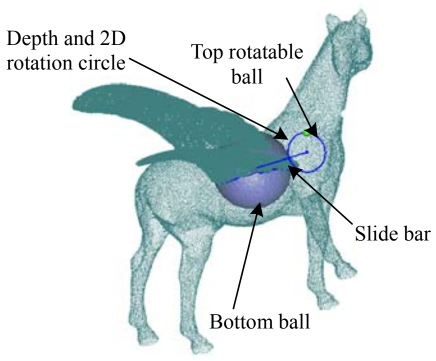


Fig.7 Local adjustment of the point position

(1) Dragging the circle along the slide bar to bring M_c to slide along the slide bar—this makes a translation along the normal of centroid.

(2) Dragging the tiny ball along the circle to bring M_c to rotate around the slide bar—this makes a rotation around the normal at centroid.

(3) Dragging the top end of the slider bar to make M_c rotate around the centroid—this makes a rotation around the centroid.

COMPUTING THE BLENDING REGION

An important step in blending point model is to calculate the blending region. After culling out two models' interlaced points, Turk and O'Brien (2002) used all points of both models as the blending region. The high computation complexity of this global blending limits its application to large scale model. We propose a local blending strategy here by constructing an implicit surface from blending regions adjacent to P_{inter} in M_c and M_p. Our local blending strategy is more efficient and not sensitive to the size of the blending model.

A simple way to find blending region might use k-d tree to find neighboring points of P_{inter}, but the blending regions identified by this way will include interlaced points, which will be impossible for RBF to generate a smooth transition region. We compute the blending regions in real-time. Once P_{inter} and M_c are transformed to new positions, blending region F_c in M_c and F_p in M_p will be calculated efficiently as follows:

(1) Fit a least square plane P_f through P_{inter}.

(2) Project P_{inter} to P_f to get a polygon P_{prj}. After scaling P_{prj} twice in different ratios using 'equidistant scale' algorithm, we get two polygons P_{in} and P_{out}, as shown in Fig.8.

$$P_k'=P_k+\boldsymbol{E}_{(k-1)k}\times\boldsymbol{N}_{\text{plane}}\cdot S,$$

where $\boldsymbol{E}_{(k-1)k}$ is the vector from P_{k-1} to P_k, $\boldsymbol{N}_{\text{plane}}$ is the normal of P_f. We use equidistant scaling since it can produce more reasonable results for both convex and concave polygons than center based scaling.

(3) Compute F_p by first projecting all points in M_p to the plane P_f, and putting all the projected points in F_p, then excluding from F_p those points on the back surface based on the depth value (Fig.9a).

(4) Compute F_c (Fig.9b): offsetting P_f as a distance along its normal to M_c to get a plane P_{front}, with points of M_c between P_f and P_{front} being put into F_c.

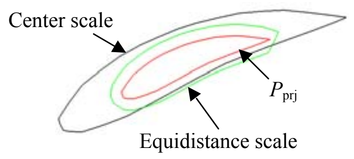


Fig.8 Center based scaling vs equal distance scaling

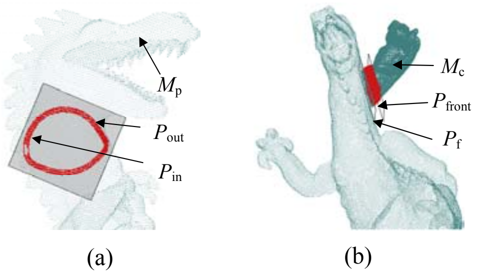


Fig.9 Blending region on M_p (a) and M_c (b)

CONSTRUCTING THE TRANSITION REGIONS

After computing the blending regions F_c and F_p, we will construct the transition region T_R, which smoothly interpolates F_c and F_p. The problem of constructing the transition region can be considered to construct a surface M passing through all distinct points in F_c and F_p, which is a classical surface reconstruction problem. A more general expression of surface reconstruction problem is: given n distinct points $\{(x_i,y_i,z_i)\}_{i=1}^{n}$ on a surface M in $\mathbb{R}^3$, find a surface M' that is a reasonable approximation to M.

In this paper, we propose to use implicit function $f(x,y,z)$ to define M. We try to find a function f which implicitly defines a surface M' and satisfies $f(x_i, y_i, z_i)=0$ ($i=1, \ldots, n$), where $\{(x_i,y_i,z_i)\}_{i=1}^{n}$ are the given points on M.

In order to avoid the trivial solution (i.e., f is zero everywhere), off-surface points are appended to the input data and are given non-zero values. We use the method in (Carr *et al.*, 2001) to generate off-surface points by projecting along the point normal. Off-surface points may be at either side of the surface. There are two problems here to solve: estimating point normal and determining the appropriate projection distance h. After constructing local triangulations for F_p and F_c, normal vectors at points can be easily and robustly computed from their triangulations. To avoid self-intersection, the projected point is taken to be the closest surface point generated, and the projection distance h is taken as the smaller distance in F_c and F_p. We assign the value 1.0 (−1.0) to the points outside (inside) of the objects.

Off-surface point gives additional constraints for implicit function f:

$$f(x_i, y_i, z_i)=d_i\neq 0,\ \ i=n+1, \ldots, N.$$

An obvious choice for f is the signed-distance function, where the d_i is chosen to be the distance to the closest surface point.

Given a set of zero-valued surface points and non-zero off-surface points, we now have a scattered data interpolation problem: we want to approximate the signed-distance function $f(x)$ by an interpolant $s(x)$. RBF is considered to be a successful solution for scattered data interpolation problem, and has been used in surface reconstruction (Savchenko *et al.*, 1995; Carr *et al.*, 2001), mesh hole-filling (Du *et al.*, 2005), mesh fusion (Jin *et al.*, 2006), and point cloud editing (Chen *et al.*, 2006) and modeling (Turk and O'Brien, 2002; Reuter *et al.*, 2003; Botsch and Kobbelt, 2005). An RBF has the form:

$$s(x)=p(x)+\sum_{i=1}^{N}\lambda_i\Phi(|x-x_i|), \qquad (1)$$

where p is a polynomial of low degree, $p=c_1+c_2x+c_3y+c_4z$. $|\cdot|$ is Euclidean norm on $\mathbb{R}^3$, $\{\lambda_i\}$ are real numbers and the base function Φ is a real-valued function on $[0,+\infty)$. Different base functions are chosen for different scattered datasets. In this paper we choose the tri-harmonic splines $\Phi(x)=|x|^3$ as the base function, as the energy minimization properties of tri-harmonic splines make them well suited to represent 3D objects. By solving a linear equation system, we can get the coefficients in Eq.(1), thus we have the expression of RBF function. Then we use a marching tetrahedral variant to sample the implicit surface. Those points obtained by the marching tetrahedral method are taken as the transition region. There are some efficient methods to solve the RBF surface (Carr *et al.*, 2001). We use a modified fast multipole method (FMM) (Beatson *et al.*, 2006) and the center reduction algorithm to improve the efficiency of solving the RBF surfaces in this paper. After centers are clustered in a hierarchical manner, far- and near-field expansions are used to generate an approximation to that part of RBF due to the centers in a particular cluster.

After the transition region T_R is computed, we can combine T_R and blend two models M_c and M_p into one model M_r (Fig.10). Since T_R is somehow overlapped with M_c and M_p, we need to eliminate the overlapped points. We first put points in M_p, whose projections on plane P_f are outside of P_{out}, into M_r. Then put all the points in M_c at the other side of P_f into M_r. Since T_R can be classified into two parts S_1 and S_2 by their distances to P_f with a threshold—S_1 being within the threshold, and S_2 outside the threshold. Those points in S_1 whose projections on P_f are inside of P_{out} will be put into M_r, and those in S_2 that lie between P_f and P_{front} will also be put into M_r.

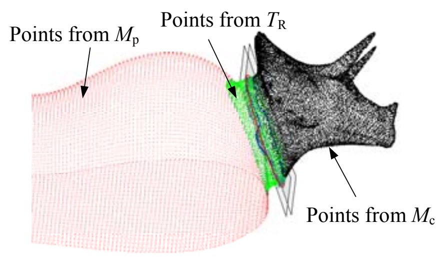
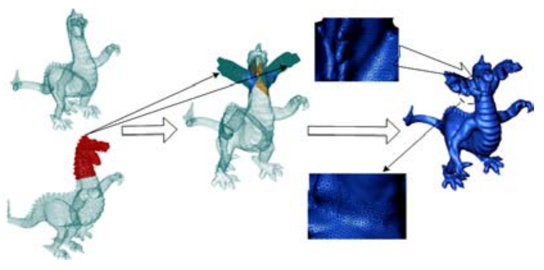


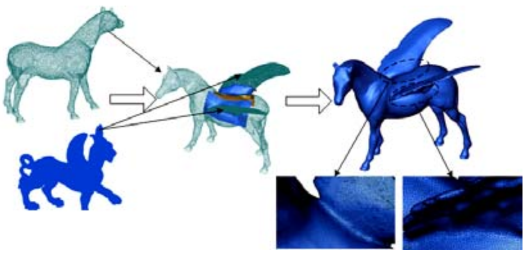
Fig.10 Combining the blending region with the original two point clouds

RESULTS AND DISCUSSION

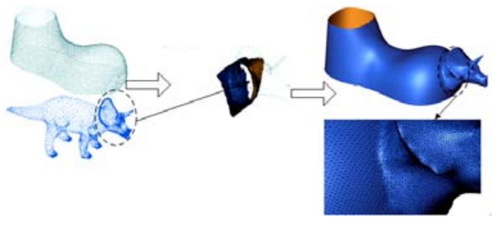
We implement our interactive blending method for point cloud on a P4 2.8 GHz with 512 MB memory PC with Windows 2000 OS. Table 1 gives a statistics on the time spent on the examples used in this paper. Due to the computation complexity of RBF, our blending system is not real-time, with the blending time ranging from several seconds to thirty seconds. Most of these point cloud models are derived from meshes by erasing edges and normal from mesh, just reserving vertices' coordinate, which make them the same as row point cloud. We construct mesh for the combining result to see the blending effect. Fig.11a and Fig.11b illustrate that we can use this interactive blending method several times to model a complicated model. Blending time includes RBF fitting time and marching tetrahedral surface sampling time. Blending of Fig.11a is much more than double the time of Fig.11b as it sampled many more points in the transition region. It shows that our blending algorithm is not sensitive to the total number of points in models, but is related to the number of points in the blending region. Fig.11c illustrates that our algorithm can blend irregular sample model (the triceratops is irregularly sampled).



(a)



(b)



(c)

Fig.11 The blending of head and body (a), horse and wings (b) and shoe and triceratops (c)

Table 1 Statistics on the time spent in the blending

Figure	Points in model	Points in blending region	RBF centers	Points in transition region	Blending time (s)
11a	75904	1512	617	16463	25
11b	76501	758	309	1191	7
11c	35002	953	337	3383	8

CONCLUSION AND FUTURE WORK

In this paper, we presented an interactive blending modeling method for point-based geometry. After intuitively and interactively dragging one point model onto the surface of another point model, the user can “cut” a portion of the model from one model, and “past” it on the other model. It is convenient for the users to find a reasonably satisfactory position for blending with the dragging and local position adjustments metaphors. Based on the polygon of interest, blending regions are automatically calculated. An implicit surface interpolating the blending regions is constructed, and a modified fast multipole method is implemented to speed up the solution and sampling of the implicit surface. As emphasized in the introduction, our algorithm is proposed to process raw point cloud. Some models used in our experiment are obtained from mesh model by erasing edges and normal, just keeping vertices coordinates, which are considered as raw point cloud. In practice, if we keep vertices’ normal, some parts of our method could be simplified:

(1) Computation of $\boldsymbol{V}_{\mathrm{T}}$ from the depth buffer. By also rasterizing the normal in the depth buffer, “costly” least squares fit can be avoided.

(2) Computing offset points for the radial basis function becomes much easier (triangulation can be avoided).

Our future work includes a GPU-based implementation to speed up the blending process and the implementation of other geometry modeling methods for point-based geometries. Another issue is the smoothness of the transition region constructed by RBF. We will provide some way to vary the smoothness of the transition region by controlling the limit of fitting functions derived from the given points in blending region.

References

Adams, B., Dutre, P., 2003a. Interactive boolean operations on surfel-bounded solids. *ACM Trans. on Graph.*, **22**(3):651-656. [doi:10.1145/882262.882320]

Adams, B., Dutre, P., 2003b. A Smoothing Operator for Boolean Operations on Surfel-bounded Solids. Geometric Computing Group Tech. Rep., Stanford University.

Adams, B., Wicke, M., Dutre, P., Gross, M., Pauly, M., Teschner, M., 2004. Interactive 3D Painting on Point-sampled Objects. Proc. Eurographics Symp. on Point

Based Graphics. Grenoble, France, p.57-66.

Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., Silva, C.T., 2001. Point Set Surfaces. Proc. Conf. on Visualization. Washington, DC, p.21-28.

Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., Silva, C.T., 2003. Computing and rendering point set surfaces. *IEEE Trans. on Visual. Computer Graph.*, **9**(1): 3-15. [doi:10.1109/TVCG.2003.1175093]

Beatson, R.K., Powell, M.J.D., Tan, A.M., 2006. Fast Evaluation of Polyharmonic Splines in Three Dimensions. Technical Report, NA2006/03. Numerical Analysis Group, Department of Applied Mathematics and Theoretical Physics, University of Cambridge.

Botsch, M., Kobbelt, L.P., 2005. Real-time shape editing using radial basis functions. *Computer Graphics Forum*, **24**(3): 611-621. [doi:10.1111/j.1467-8659.2005.00886.x]

Carr, J.C., Beatson, R.K., Cherrie, J.B., Mitchell, T.J., Fright, W.R., Mccallum, B.C., Evans, T.R., 2001. Reconstruction and Representation of 3D Objects with Radial Basis Functions. Proc. SIGGRAPH. Los Angeles, CA, p.67-76. [doi:10.1145/383259.383266]

Chen, F.Z., Chen, Z.Y., Ding, Z., Ye, X.Z., Zhang, S.Y., 2006. Filling holes in point cloud with radial basis function. *J. Computer-Aided Design and Computer Graphics*, **18**(9): 1414-1419 (in Chinese).

Du, J., Zhang, L.Y., Wang, H.T., Liu, S.L., 2005. Hole repairing in triangular meshes based on radial basis function. *J. Computer-Aided Design and Computer Graphics*, **17**(9):1976-1982 (in Chinese).

Fang, S.F., Srinivasan, R., Raghavan, R., Richtsmeier, J.T., 2000. Volume morphing and rendering—an integrated approach. *Computer Aided Geometric Design*, **17**(1): 59-81. [doi:10.1016/S0167-8396(99)00039-4]

Grossman, J.P., Dally, J., 1998. Point Sample Rendering. Proc. 9th Eurographics Workshop on Rendering. Vienna, p.181-192.

Guo, X.H., Li, X., Bao, Y.F., Gu, X.F., Qin, H., 2006. Meshless thin-shell simulation based on global conformal parameterization. *IEEE Trans. on Visual. Computer Graph.*, **12**(3):375-385. [doi:10.1109/TVCG.2006.52]

Igarashi, T., Hughes, J.F., 2003. Smooth Meshes for Sketch-based Freeform Modeling. Proc. Symp. on Interactive 3D Graphics. Monterey, CA, p.139-142. [doi:10.1145/641480.641507]

Jin, X.G., Lin, J.C., Wang, C., Feng, J.Q., Sun, H.Q., 2006. Mesh fusion using functional blending on topologically incompatible sections. *The Visual Computer*, **22**(4):266-275. [doi:10.1007/s00371-006-0004-8]

Levin, D., 2003. Mesh-independent Surface Interpolation. Geometric Modeling for Scientific Visualization. Springer-Verlag, p.37-49.

Liu, Y.S., Zhang, H., Yong, J.H., Yu, P.Q., Sun, J.G., 2005. Mesh blending. *The Visual Computer*, **21**(11):915-927. [doi:10.1007/s00371-005-0306-2]

Pauly, M., Keiser, R., Kobbelt, L.P., Gross, M., 2003. Shape Modeling with Point-sampled Geometry. SIGGRAPH. San Diego, CA, p.641-650. [doi:10.1145/1201775.882319]

Qian, J.F., Chen, Z.Y., Zhang, S.Y., Ye, X.Z., 2005. The detection of boundary point of point cloud compression. *J. Image Graph.*, **10**(2):21-28 (in Chinese).

Reuter, P., Tobor, I., Schlick, C., Dedieu, S., 2003. Point-based Modelling and Rendering Using Radial Basis Functions. GRAPHITE. Melbourne, Australia, p.111-118. [doi:10.1145/604471.604494]

Savchenko, V., Pasko, A., Okunev, O., Kunii, T., 1995. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum*, **14**(4):181-188. [doi:10.1111/1467-8659.1440181]

Turk, G., O'Brien, J.F., 2002. Modelling with implicit surfaces that interpolate. *ACM Trans. on Graph.*, **21**(4):855-873. [doi:10.1145/571647.571650]

Yang, Z.Y., Zheng, W.T., Peng, Q.S., 2005. Interactive boolean operations on general point models. *J. Computer-Aided Design and Computer Graphics*, **17**(5):954-961 (in Chinese).

Zwicker, M., Pauly, M., Knoll, O., Gross, M., 2002. Pointshop 3D: an interactive system for point-based surface editing. *ACM Trans. on Graph.*, **21**(3):322-329. [doi:10.1145/566654.566584]