1993

Distributed Modeling and Rendering of Splines Using Ganith and SplineX

Chandrajit L. Bajaj

Jindon Chen

Susan B. Evans

Report Number:
93-006

DISTRIBUTED MODELING AND RENDERING OF SPLINES USING GANITH AND SPLINEX

Chandrajit L. Bajaj
Jindon Chen
Susan B. Evans

CSD-TR-93-006
January 1993

Distributed Modeling and Rendering of Splines Using Ganith and SplineX[†]

Chandrajit L. Bajaj *Jindon Chen* *Susan B. Evans*

Department of Computer Sciences
Purdue University
(Tel)317-494-6531
(Fax)317-494-0739
{bajaj, jdc, evans}@cs.purdue.edu
West Lafayette, IN 47907

Abstract

There are two main contributions of this paper. One is the description of the architecture of the SplineX Client/Server toolkit for interactive modeling and display of algebraic curves and surface splines over a Bernstein-Bezier(BB) basis. The second is our enhancements and experiments with the implementation of modeling and rendering algorithm dealing with algebraic surface splines. In particular, we describe the implementation of distributing the rendering process of a spline consisting of several Bezier simplices over a network of workstations. Furthermore, we describe the use of SplineX for the interactive control and surface selection of fitting surfaces.

Keywords: Distributed graphics, curve and surface modeling, graphics software architectures, interactive techniques

1 Introduction

A conventional modeling tools is a single user/single machine environment. However, such an environment has many disadvantages. Working in a sequential maner, a conventional modeling tool may take an long time to unpleasantly overcome a massive computation task, which is common in non-trivial modeling or rendering applications. Another disadvantage is that the size of the system may become huge and too large for the host as functionality

[†]Research in part supported by NSF grant CCR 90-02228, NSF grant DMS 91-01424 and AFOSR contract 91-0276.

is expanded. Also, it is not easy to extend the system as all the existing codes reside in one single program. Collaboration of multiple users is out of question in a single user environment. Recently, distributed graphics system has been more and more turning into a trend. A distributed system consists of sibling subsystems distributed over a network of machines. Massive computation may be decomposed and carried out on different machine so as to speed up the processing by exploit the parallelism of the computation task. Connecting with each other over the network, each of the sibling system is specialized on its own functionality. Thus the size of each subsystem can be kept small. There are also high extensibility in the sense that a subsystem can be developed independently and to the existing system hooked up by network connection. Collaboration is also achievable in a multiple user multiple machine environment.

Developed in the Computer Sciences Department, Purdue University, Shastra [ABR91], of which Ganith and SplineX are components, is a highly extensible, distributed and collaborative geometric software environment consisting of a growing set of individually powerful and interoperable (client-server) which support collaborative design sessions. In the Shastra environment multiple users (say, a collaborative engineering design team) interactively create, share, manipulate, simulate and visualize complex geometric designs over a heterogeneous network of workstations and supercomputers. Currently, Shastra consists of Ganith [BR91], Shilp [ABI+91], Vaidak [BBF91], Bhautik [BOS92], Gatti [BC92] and SplineX.

Algebraic curves and surfaces in implicit or parametric form are one of the common modeling tools in modeling environments. Shastra system, as well as Ganith and SplineX, is based on this tools. They are thus our objects to be distributed. Decomposition, one of the most common and efficient way of computing an implicit curve or surface, has indicated

the feasibility of distributed computation over them. Piecewise curves and surface also suggest their parallelism by their definition.

There are two main contributions of this paper. One is the description of the architecture of the SplineX Client/Server toolkit for interactive modeling and display of algebraic curves and surface splines over a Bernstein-Bezier(BB) basis. The second is our enhancements and experiments with the implementation of modeling and rendering algorithm dealing with algebraic surface splines. In particular, we describe the implementation of distributing the rendering process of a spline consisting of several Bezier simplices over a network of workstations. Furthermore, we describe the use of SplineX for the interactive control and surface selection of fitting surfaces.

The rest of the paper is organized as follows. Section 2 presents an overview of the SplineX system, the Ganith system and the connection between them. Section 3 discusses the distributed rendering in terms of system aspects, implicit Bernstein patches and parametric Bernstein patches. Section 4 addresses distributed modeling. Section 5 concludes the paper.

2 Overview of SplineX and Ganith

SplineX and Ganith are X-11 [SG86] and XS [ABB+91] based application toolkits under the umbrella of Shastra [ABR91].

The SplineX toolkit manipulates different geometric patches in Bernstein-Bezier basis. One can model a geometric object by BB based patches which are continuous to each other. Figure 1 shows a degree 4 surface(finger) displayed in 8 tetrahedral domains.

The Ganith algebraic geometry toolkit [BR91] manipulates arbitrary degree polyno-

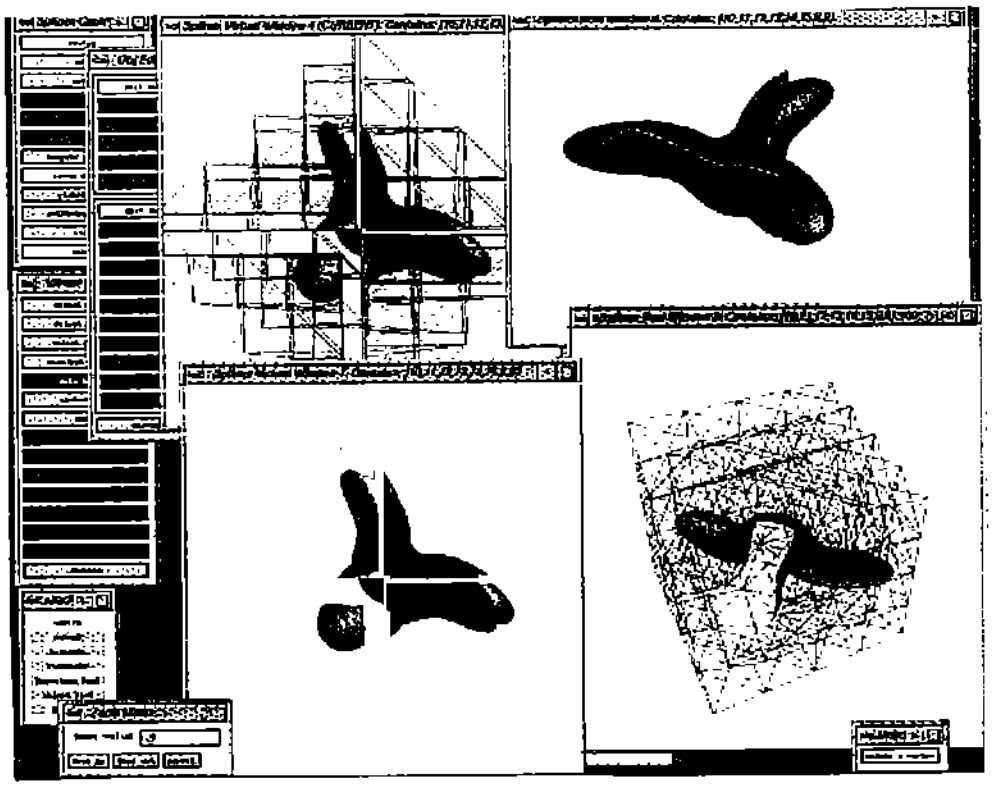

Figure 1: Display of a degree 4 surface (Finger) in SplineX

mials and power series. It can be used to solve a system of algebraic equations and visualize its multiple solutions. Example applications of this for geometric modeling and computer graphics are curve and surface display, curve-curve intersections, surface-surface intersections, global and local parameterizations, implicitizations, and inversions. It also incorporates techniques for multivariate interpolation and least-squares approximation to an arbitrary collection of points and curves.

2.1 Mathematical foundation of SplineX

The objects manipulated in SplineX are geometric patches, which are represented implicitly or parametrically in Bernstein-Bezier basis. A *simplex* in $\Re^n$ is the convex hull of $n+1$ points that are not contained in any $n-1$ dimensional hyperplane. Let F be a polynomial

of degree k in Bernstein-Bezier basis representation defined over a simplex S.

$$F(\alpha_0, ..., \alpha_1) = \sum_{\substack{i_0, ..., i_n \geq 0 \\ i_0 + ... + i_n = 0}} b_{i_0,...,i_n} B^k_{i_0,...,i_n}(\alpha_0, ..., \alpha_n),$$

where

$$B^k_{i_0,...,i_n} = \frac{k!}{i_0!, ..., i_n!} \alpha_0^{i_0}, ..., \alpha_n^{i_n}.$$

Mathematically, $F(\alpha_0, ..., \alpha_1) = 0$, an implicit patch in Bernstein-Bezier basis is defined by its dimension, degree, domain simplex, and the barycentric coordinates over this domain [BBB] [Far90] [Far86] [dB87]. In particular, a 3D patches is defined by a tetrahedral domain simplex, and barycentric coordinates of certain degree. An parametric surface is represented as follows,

$$x = \frac{F(s,t)}{D(s,t)}, \qquad y = \frac{G(s,t)}{D(s,t)}, \qquad z = \frac{H(s,t)}{D(s,t)},$$

where $F(s,t)$, $G(s,t)$, $H(s,t)$ and $D(s,t)$ are polynomials in Bernstein-Bezier basis.

Currently, SplineX handles both 2D and 3D cases.

In SplineX, a bounding box is applied for an implicit or parametric surface. In implicit cases, the domain simplex is the bounding box, where in parametric cases, a cubical box is given as the bounding box.

2.2 System Architecture of SplineX

The SplineX system is controlled by a multi-window graphical user interface system, which provides convenient method to manipulate different subsystems. However, SplineX itself, as a whole, is a subsystem in the Shastra system[ABR91], a distributed collaborative geometric modeling environment. The network connection subsystem connects SplineX with other Shastra subsystems.

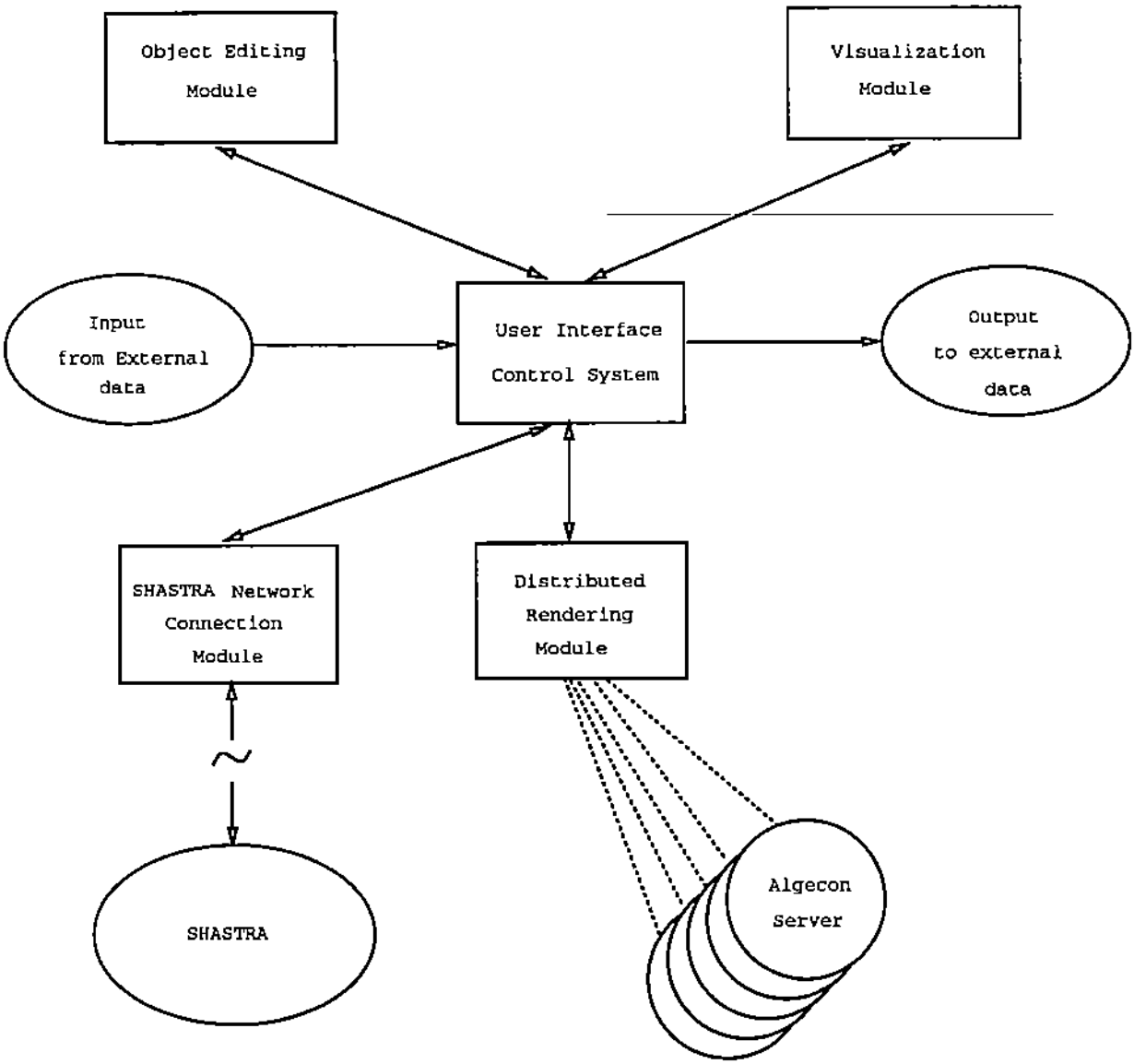

Figure 2: Architecture of SplineX

As shown in Figure 2, the SplineX toolkit has several components. Object editing module, visualization module, input module, output module, Shastra connection module, distributed rendering module, organized by the user interface control system, form the front end of the system. The massive computation server subsystem, or Algecon servers, forms the back end. The front end resides in one process, while the back end is a collection of homogeneous servers that run on various hosts. The distributed rendering module dispatches computation tasks to algecon servers through asynchronous remote procedure calls. A server, in turn, can response to more than one SplineX front end. The system currently runs under SunOS Release 4.1.1 on Sun workstations and IRIX Release 4.0.5 System V on SGI workstations.

The Algecon server subsystem consists of three portions, the network interface, the computational core and border justification module. The computational core of the Algecon server is the Algecon library. Developed by Doug Moore and Joe Warren in Rice University, the Algecon library is an adaptive contour generator for multivariate polynomials in the Bernstein-Bezier basis. Algecon library takes domain simplices and polynomials in barycentric coordinates, decompose the patches inside the simplices, returns triangular meshes which approximates the surface patches to the desired extend. Algecon library is able to decompose a surface patch inside a space region which consists of simplices. The surface patch can be either defined by one single polynomial or by piecewise polynomials. User can choose decomposition parameters such as maximum and minimum levels for adaptive subdivision, maximum error tolerated in less-than-completely-refined triangle, number of extra thicknesses of tetrahedra generated, size of vertex figures in hybrid method for surface triangulation. The network interface communicates with the front ends of SplineX,

converting the data from network representation to that of Algecon library. The justification module justifies the border of the resulting patch such that it lies on the boundary of the simplex.

The user interface control system is responsible for the interaction between the user and different components. It could be further subdivided into a user interface and a controller. Once a command is entered and edited to satisfaction, the controller sends it to the proper component for execution. During the execution, the user interface may prompt the user for further input upon the request from the executing component. The controller is also responsible for the interactions between the components which are without human interference.

The object editing module provides the functionality of creating, merging, moving, rotating, scaling, duplicating and destroying an object, moving, inserting, deleting a vertex of the objects, or updating the coordinates of an object. The visualization module is responsible of the display of the object on the screen. Utilizing the XS[ABB+91] routines, it draws the objects in various projections and shading models. It displays also the cross sections in certain directions of an object for better human-machine interaction. As part of Shastra [ABR91], SplineX is connected to other parts of Shastra via the Shastra network connection module, which will be discussed later. The distributed rendering module subdivides and renders the decomposition the surface patches among the Algecon servers and sews the resulting patches together to form a whole piece of surface. It also calls Algecon servers to convert Bernstein representation of a surface between different domains. The distributed rendering system will be described at length in Section 3.

2.3 Client-Server

In the Shastra environment, every components run as independent processes on separate workstations having separate user interfaces (using X-11 and Motif). The application toolkits make use of a designed network library to communicate data structures conveniently between each other and manage multiple connections across a network. The network library is designed around the highly extensible `client-server` paradigm and utilizes TCP/IP. Each application runs as a server for the functionality it offers. and as a client capable of requesting functionality from other sibling systems. Applications maintain multiple concurrent connections to other applications on multiple hosts. this is effected using the multiplexing facility accorded by the "select" system call. The application dynamically opens connections to different systems and registers handlers with the multiplexing layer or closes such connections. In server mode, the application sets up shop at a well known port, and awaits requests for connections from other systems. In client mode, the application attempts to connect to a well known port for a specific service. Both of these actions can be performed identically in all systems.

The upper part of data communication between SplineX and Ganith is based on XDR protocols. The interformation of a Ganith graphic object is packed into a well designed XDR data structure and sent over to SplineX, while the later unpacks it and transform it to its own data format, and vice versa. For example, in Figure 3, a degree 4 surface patch joins 3 cylinders smoothly. The picture on the right is displayed by Ganith while the one on the left is by SplineX after the data is sent from Ganith to SplineX.

In Section 4, we use this connection to do distributed design.

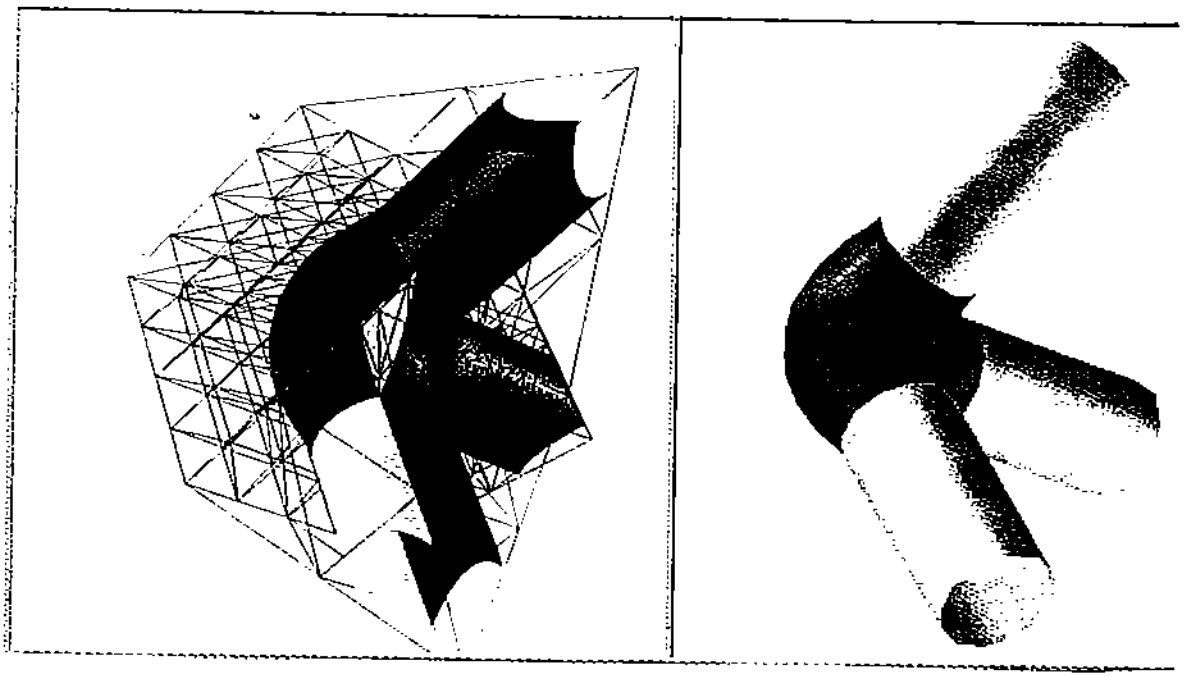

Figure 3: Connection between SplineX and Ganith

3 Distributed Rendering

The distributed rendering module serves as subdividing and rendering the decomposition of surface patches among the Algecon servers and sewing up the resulting patches by removing the numerical inaccuracy along the borders between the patches.

3.1 Distributed rendering of implicit patches

One of design issues in interative modeling system is quick response for user commands. However, a call to Algecon library is rather time consuming, especially when a collection of patches over different domains are computed in a serial manner.

Therefore we design the distributed rendering module and Algecon server in an effort to speed up the decomposition of the surface patches. The distributed rendering module subdivides the objects and dispatch them to Algecon servers on different machines while the latters carry out the computation in a parallel fashion. Each of the server computes a

part of the whole surface patch.

However, there is an issue about how fine the subdivision should be. Here we discuss two fashions of doing the subdivision. The first one is that a unit of the subdivision is a single simplex. Hence one server may serves more than one decomposition service request. Another way is that a unit consists more than one simplex and the number of the units is decided by the number of Algecon servers available and hence one server serves one unit. The first fashion is simple and it can also utilize the concurrency inside one host. However its disadvantages are the overhead of networking communication and the extra effort to sew up the small pieces of patches. The second one, however, requires a more complicated scheme to divide a region. We choose the first one for the sake of simplicity.

After patches are sent back from Algecon servers, they need to put together to form a whole piece. If two patches are joined to each other, they share a common boundary, which is approximated by a polyline. However, numerical inaccuracy and the fact that polyline boundary is generated independently to other simplices may bring a gap between the two patches. The picture on the left of Figure 4 shows us a gaps between the pieces of a degree 4 surface. Therefore, the task of putting the pieces together is mainly to bring the boundaries of two neighboring pieces together.

In 3D case, a piece of boundary of a surface patch is actually a 2 dimensional curve defined over a triangular face, along with the weights on this face, of the domain tetrahedron. Similarly, the end points of a boundary curve is a 1 dimensional point defined over an edge, along with the weights on this edge, of the domain tetrahedron. Based on this observation, our scheme works as follows, Within one server call, extra call to the Algecon library is made with the same decomposition parameters to compute the points on each edge and

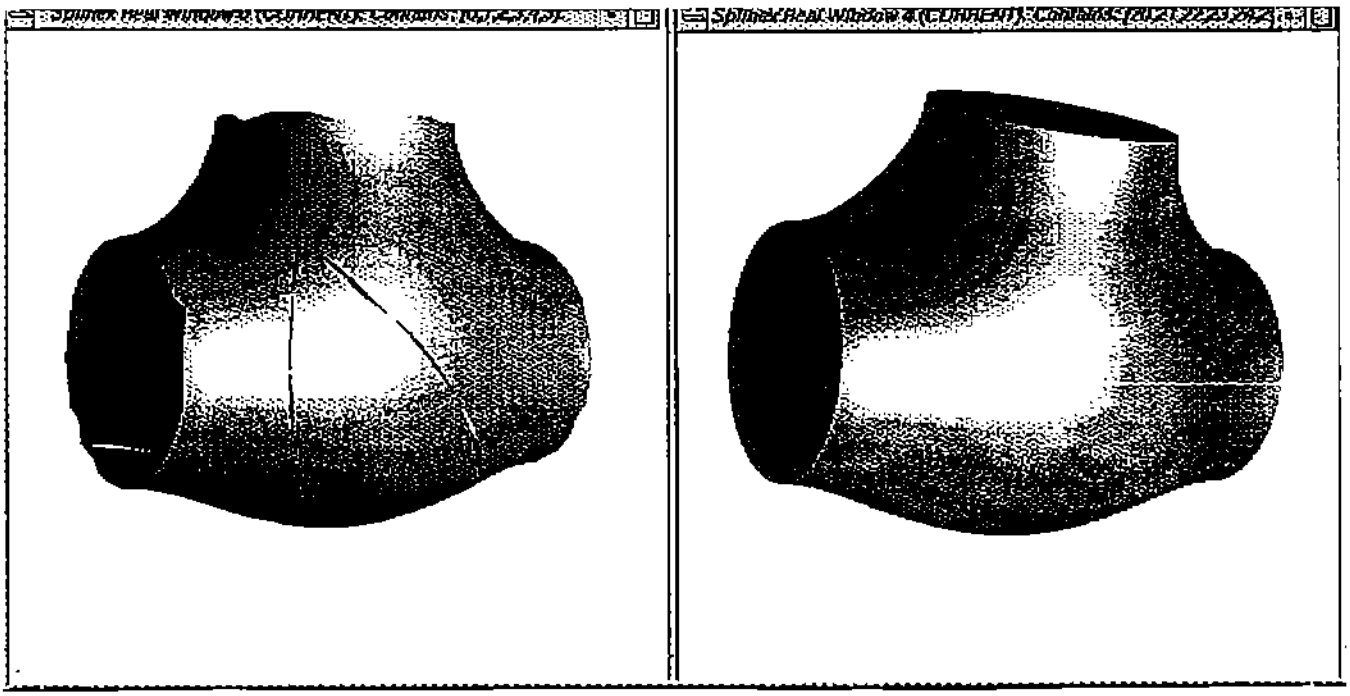

Figure 4: before and after merging several pieces of a degree 4 surface

curves on each face of the domain tetrahedron. Then we adjust the curves to its endpoints and adjust the boundary of the patch to these curves. Please notice that as there maybe more than one piece of patch inside the domain, there maybe more than one piece of curve on the boundary.

When the server calls return, neighboring patches are merged by pairing their boundary polyline by their end points and merging them. If there are more than a pair of curves with the same pair of endpoints, choose extra points on the curves untill they are distinguished. When merging two pieces of boundaries, the sparser one is mapped to the other.

If two neighboring domain tetrahedra have the same weights on the triangular face they merge each other, their boundaries on this face are exactly the same. Therefore, boundaries of both patches are adjusted to the common polyline curve. Replace each vertex on the sparser polyline by the closest point on the denser polyline and replace each edge of the sparser polyline by the vertex sequence of the denser polyline that is between the vertices

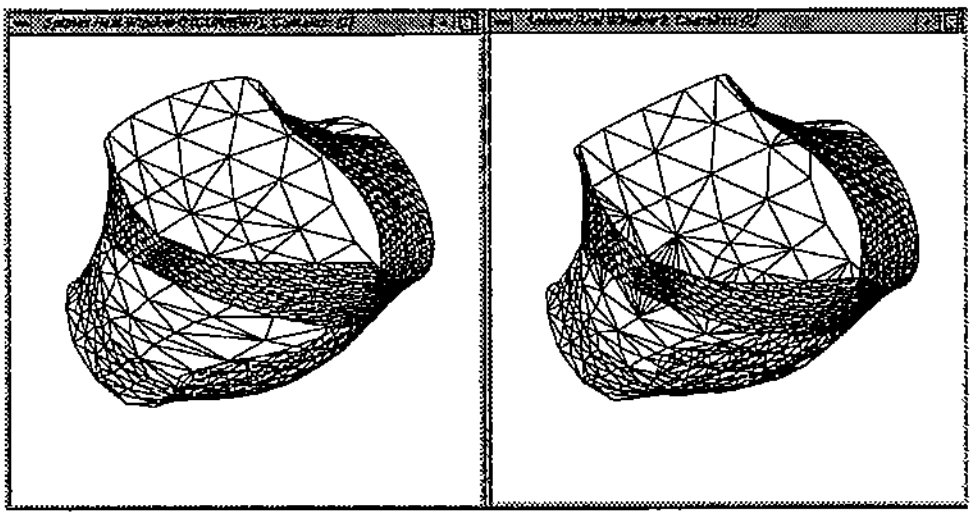

Figure 5: Merging of patch boundaries

which replace the end point of this edge. The triangular facets besides the boundary of the spacer side are subdivided to accommodate the change. See Figure 5.

Figure 4 shows the cases before and after of merging six pieces of a degree 4 surface.

Please notice that as the matching polylines are similar to each other, linear time is enough to merge them. Therefore, it is easy to show this algorithm runs in time of $O(n)$, where n is the number of vertices on the boundary.

3.2 Distributed rendering of parametric patches

Algecon, although originally designed for computation of implicit patches, can also be used to generate parametric surface patches. Distributed rendering makes it even more feasible and attractive.

For the reason of simplicity, we only consider a special form of rational parametric

surface equation as follows.

$$x = \frac{F(s,t)}{D(s,t)}, \qquad y = \frac{G(s,t)}{D(s,t)}, \qquad z = \frac{H(s,t)}{D(s,t)}, \tag{1}$$

where $s \in [0,1], t \in [0,1]$ and $D(s,t) \neq 0$ over the domain, And we assume that what of interests is only the surface patch within the unit bounding cubic where $x \in [0,1]$, $y \in [0,1]$, $z \in [0,1]$. Obviously, an arbitrary rational surface equation with an arbitrary bounding box can be mapped into this form easily.

Consider equations (1), they are implicit surface patches, denoted as P_{stx}, P_{sty} and P_{stz} respectively, over $S-T-X$, $S-T-Y$ and $S-T-Z$ spaces. We can view them as projections of an object P in a 5 dimensional space $S-T-X-Y-Z$ to 3 dimensional spaces. So if we can construct P in 5 dimensional space $S-T-X-Y-Z$, its projection to $X-Y-Z$ space, denoted as P_{xyz} is the desired patch.

Since the surface is a parametric surface, so, defined by equations (1), there is a one to one correspondence between P_{xyz} and P_{st}, where P_{st} is the projection of P to $S-T$ space. This also means that the projection mappings from P to P_{xyz}, from P_{stx} to P_{st}, from P_{sty} to P_{st}, from P_{stz} to P_{st} are one to one correspondent.

We present the following scheme to generate the parametric patches inside the cubical bounding box.

(1) (i)Consider surface $F(s,t) - xD(s,t) = 0$, where $x, s, t \in [0,1]$ as its bounding cubic. Break the bounding cubic into 6 tetrahedra as shown in Figure 6. Make 6 remote calls to algecon server to convert the equation into 6 barycentrically represented equations over the 6 tetrahedra. Make 6 remote calls to algecon server, each for one of the 6 tetrahedra, to compute the surface patches inside them. Then, like we have discussed in the previous

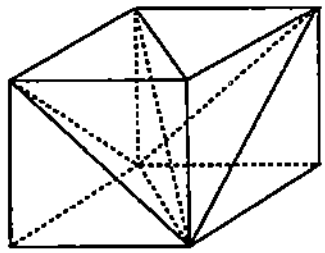

Figure 6: Tetrahedral subdivision of a cubical bounding box

section, the patches are sewn together to form a whole patch P_x (actually it could be several pieces) represented by triangular mesh inside the bounding cubic of $x, s, t \in [0, 1]$.

(ii) Do the same as (i) to surface $G(s,t) - yD(s,t) = 0$, where $y, s, t \in [0, 1]$ and, as the result, we get patch P_y.

(iii) Do the same as (i) to surface $H(s,t) - zD(s,t) = 0$, where $z, s, t \in [0, 1]$ and, as the result, we get patch P_z.

(2) Build patch P_{xyz} in $X - Y - Z$ space out of patches P_x, P_y and P_z.

We first project triangular meshes P_x, P_y and P_z onto $S - T$ plane, denoted respectively as P'_x, P'_y and P'_z. Now P'_x, P'_y and P'_z are overlapping each other on $S - T$ plane. as the projection mappings are one to one correspondent, for any vertex (s,t) on P'_x, there exists one and only one vertex (s,t,x) on P_x. Same to vertices on P'_y and P'_z.

Please notice that as we only compute the portion inside the bounding box. The projections of P'_x, P'_y and P'_z may not coincide with each other. However only the common portion of P'_x, P'_y and P'_z represents the patch P_{xyz} inside the bounding box. Thus, before going any further, we need to compute the intersection of P'_x, P'_y and P'_z and cut off the differences which are irrelevant to P_{xyz}. This can be done as follows.

(a) Computing the borders of P'_x, P'_y and P'_z, respectively, $Bd(P'_x)$, $Bd(P'_y)$ and $Bd(P'_z)$,

each of which is a simple polygon.

(b) Perform a polygon intersection operation on the border polygons.

$$R = Bd(P'_x) \cap Bd(P'_y) \cap Bd(P'_z).$$

(c) Cut P'_x according to the intersection R, Denote the remain region as P''_x. As we will see, in P''_x, it is sufficient to store only the vertices that are inside R. Compute P''_y and P''_z out of P'_y and P'_z in the same way.

(a) and (c) take $O(n)$ time while (b) takes $O((n+k)\log n)$ time, where k is the number of intersections between the edges of the polygons. As real original curves other than approximating polylines are considered, $Bd(P'_x)$, $Bd(P'_y)$ and $Bd(P'_z)$ consist of segments of $F(s,t) - D(s,t) = 0$, $G(s,t) - D(s,t) = 0$, $H(s,t) - D(s,t) = 0$ inside bounding box $\{x = 0,1; y = 0,1\}$ and the edge of the bounding box. Hence the number of their intersection points is $O(d_1 d_2)$, where d_1, d_2 are the two largest degree of those of $F(s,t) - D(s,t) = 0$, $G(s,t) - D(s,t) = 0$ and $H(s,t) - D(s,t) = 0$. We assume that polylines $Bd(P'_x)$, $Bd(P'_y)$ and $Bd(P'_z)$ are good approximation to the curves such that they intersect with each other and the bounding box in the same number. Hence the number of vertices on P''_x, P''_y and P''_z is now $O(n + d_1 d_2)$. Assuming that a good approximation goes with that $n > d_1 d_2$, the number of vertices remains $O(n)$. Similar to implicit surface, there may be more than one separated pieces of parametric surface P inside the bounding box, depending on how the bounding box is chosen.

Now we do a union of the vertex sets of P''_x, P''_y, P''_z and R.

$$V = P''_x \cup P''_y \cup P''_z$$

Then for each connected component of R, we perform a triangulation such that the vertices

of the subdivision are elements of V. The triangulation takes $O(n \log n)$ time, where n is the cardinal number of V. Let's denote the resulting triangulation as T_{st}.

Then for each vertex $(s,t) \in V$, we compute the corresponding (x,y,z) value as follows. We locate the triangular facets, denoted as $\triangle_x$ on P_x, whose projection on P'_x contains (s,t). As each vertex of the patches has a norm vector associated with it, we do a conic interpolation on $\triangle_x$ with respect to (s,t), whose result is the x value. Similarly, we get y and z. Actually, if $(s,t) \in P''_x$, then (s,t) is the projection of a vertex (s,t,x) on P_x, the x value is just taken without interpolation. There are known planar point location algorithms which run in $O(n)$ or $O(n \log n)$ time and space for preprocessing and $O(\log n)$ time for a query. [Col86] [EGS86] [ST86] Thus $O(n \log n)$ time is sufficient for locating all the vertices.

Augmenting each vertex $(s,t) \in V$ to be (s,t,x,y,z) by the x,y,z value we get above, we construct the vertex set in 5 dimensional space. Therefore, subdivision T_{st} is augmented to be T, a subdivision of P on 5 dimensional space. Projecting T to 3 3 dimensional space $X-Y-Z$, we get P_{xyz}.

So after all, part (2) of our algorithm runs in $O(n \log n)$ time, where n is the number of vertices of on P_x, P_y and P_z.

Please notice that in step (1) of the algorithm, for a piece of parametric patch inside a bounding box, we call Algecon server 18 times for converting the Bernstein-Bezier basis between different domains, and 18 times for computing the triangular facet representation of the surface patches in different domains. The computation is massive and would take a long time. Fortunately, with the distributed rendering subsystem, all this Algecon calls are dispatched to different machines across the network. Therefore, for a piecewise parametric surface, there are two levels of distributed rendering. The first level is, as what we have

discussed in the previous section, the distribution of different pieces of an implicit surface or an equation. The second level distributes the decomposition of different equations of the the parametric surface to different machines.

3.3 System aspects of the distributed rendering module

The distribution of computation over a network environment also brings up some issues of system aspects.

Sun remote procedure call mechanism is utilized for the connection between a SplineX front end and an Algecon server. One can start or delete an Algecon server on a specified host through a user interaction in a SplineX front end. An entry in the server map of the distributed rendering module indicates the connection with the server. When a decomposition task is requested, the distributed rendering module dispatches the data of each object to different machines in a round robin fashion. The Algecon server sends back the resulting patch to the front end. The client-server interaction is in an asynchronous way such that the front end does not wait for the result from the server before going further. When the resulting data is sent back to the client, a registered callback function is invoked to process the data. On the other side, the server performs the computation task of different objects concurrently. Receiving a service request from a front end, the server forks a child process to carry out the computation and communicate with the front end while the parent process continue to wait for other request from front ends.

Storing the objects in an array of structure, SplineX provides a nice architecture for asynchronous call. Each object has a unique identification number, which is associated to asynchronous call so that the callback function is able to figure out whose data it is when

the call is returned.

The resulting data from the server may not be consistent with other data in the client as some changes may have happened between an asynchronous call is made and the result is back. Therefore, before an asynchronous call is returned, all operations on the object that may cause inconsistency are recorded and combined. Upon the return of the call, the callback function applies the changes to the resulting data to maintain the consistency.

Each object has a version number, or a time stamp. Each time an asynchronous call is made, the version number is increased by one and associated with the call. Upon return, the result is discarded if the version numbers are inconsistent. Therefore, a call to the same object always overwrites previous ones. We further more keep in the record of an object that the host, and the process number of the serving child process, so that we are able to terminate an asynchronous call once it is found obsolete. However, authentication check should be made in the server side, or one can terminate someone else's service.

The distributed rendering module periodically sends echo request to all the servers on its server map. If it has not received from a server for a certain times, it assumes that the server is gone and remove the entry from the server map. On the other side, If an Algecon server has not received echo requests from anyone for a certain a mount of time, it terminates itself.

The Algecon servers are invoked by `rsh` system call, with the help of `set uid` mechanism. The distributed rendering module locally invokes a C program, *startserv*, which has its uid set. *Startserv*, which now acts like the owner of the program, in turn execute a *rsh* command to invoke an Algecon server on the specified host. This scheme is convenient. It does not even require the user to have access right to the machine on which the Algecon

servers are to be run. In a local network that has a global network file system, `startserv` updates the `.rhosts` file of the owner of the program so that the front end can be run on any host.

4 Distributed Surface Modeling

4.1 Distributed Design

Distributed surface modeling using Ganith and SplineX provides interactive control and surface selection of fitting surfaces. Ganith is used to create families of C^k fitting surfaces, and SplineX is used to control the shape of the surface and interactively select a suitable fit from a C^k family in an intuitive way.

A family of C^k fitting surfaces may be created in Ganith using Hermite interpolation [BI92]. Hermite interpolation characterizes, in terms of the nullspace of a matrix, the structure of a family of all algebraic surfaces that contain, with C^k continuity, a given collection of data points and space curves (defined implicitly as the common intersection of algebraic surfaces or in rational parametric form) possibly having associated normal directions. The result of Hermite interpolation is a q parameter family of algebraic surfaces $f(x, y, z) = 0$ of a given degree that satisfy given geometric properties. The equation of the family has the generic form

$$f(x,y,z) = \sum_{i=0}^{n} \sum_{j=0}^{n-i} \sum_{k=0}^{n-i-j} c_{ijk} \cdot x^i y^j z^k = 0, \tag{2}$$

where each c_{ijk} is a homogeneous linear combination of q-parameters $r_1, r_2, \cdots, r_q$.

Remote procedure calls are used to transfer the families of fitting surfaces from Ganith to SplineX. Figure 7 shows an instance of a quartic family of surfaces for blending 3 cylinders in Ganith and the same surface after the family has been transfered to SplineX for interactive

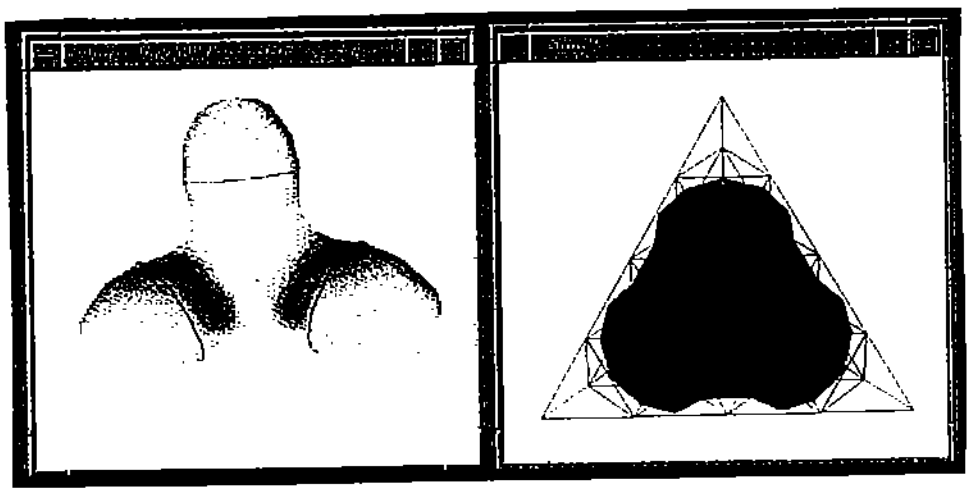

Figure 7: Distributed modeling

shape control. Distributed design is also used for converting polynomials from the standard power basis to Bernstein form over an arbitrary tetrahedron. To take advantage of the symbolic manipulations provided by the Lisp process of Ganith, we let Ganith function as a server for converting from power basis to Bernstein form. SplineX then acts as a client and makes remote procedure calls to Ganith to convert families of surfaces to Bernstein form.

Transferring a family of surfaces to the Bernstein-Bezier (BB) basis and changing Bernstein weights in SplineX provides an easy way to manipulate a family. The weights for a family of surfaces will be a linear combination of the free parameters of the surface. By changing the weights in SplineX we can interactively control the shape of the surface, and by keeping a system of linear equations derived from the weights consistent we can ensure that the selected surface is an instance of the family. The next section describes the algorithm in more detail.

4.2 Interactive Shape Control of Interpolating Surfaces

Bajaj et al.[BIW91] proposed least squares approximation to select an initial instance surface from the family obtained from Hermite interpolation. Alternately, initial default values

for the free parameters may be chosen using least squares approximation to some natural surface, such as a sphere or ellipsoid. Even though we can get some geometric intuition from least squares approximation, we may want to change the shape of the computed surface interactively by modifying the values of the free parameters. However, since the computed surface $f(x, y, z) = 0$ is a polynomial in the standard power basis, its coefficients are algebraic, not geometric. That is, they contain little intuitive geometric information, hence they do not provide a convenient tool with which the shape of an algebraic surface can be controlled intuitively.

Sederberg [Sed85] presented an idea in which free form piecewise algebraic surface patches defined in trivariate barycentric coordinates using a reference tetrahedron and a regular lattice of control points imposed on the tetrahedron. The coefficients of a surface defined in this way are assigned to the control points, and there is a meaningful relationship between the coefficients and the shape of the surface.

The essence of his idea is to consider an algebraic surface $f(x, y, z) = 0$ as the zero contour of the trivariate function $w = f(x, y, z)$. Note that the surface equation of the family of Hermite interpolating algebraic surfaces contains q free variables r_i in its coefficients. A specific portion of a surface can be selected for shape control by defining a tetrahedron which encloses that portion. Given a tetrahedron, the polynomial $f(x, y, z)$ in power basis can be symbolically converted into a polynomial $F(s, t, u)$ in barycentric coordinates, defined with respect to the tetrahedron.

Let a tetrahedron be specified by the four noncoplanar vertices P_{n00}, P_{0n0}, P_{00n}, and P_{000}. Then, the coordinates $P = (x, y, z)$ of a point inside the tetrahedron are related to the barycentric coordinates (s, t, u) by $P = sP_{n00} + tP_{0n0} + uP_{00n} + (1 - s - t - u)P_{000}$,

$s, u, t, (1-s-t-u) > 0$. Control points on the tetrahedron are defined by $P_{ijk} = \frac{i}{n}P_{n00} + \frac{j}{n}P_{0n0} + \frac{k}{n}P_{00n} + \frac{n-i-j-k}{n}P_{000}$ for nonnegative integers i, j, k such that $i+j+k \leq n$. Each control point is associated with a weight w_{ijk}, which is a linear combination of τ_i, $i = 1, 2, \cdots, q$. All these together define the q-parameter algebraic surface family in barycentric coordinates,

$$F(s,t,u) = \sum_{i=0}^{n}\sum_{j=0}^{n-i}\sum_{k=0}^{n-i-j} w_{ijk} \cdot \binom{n}{i,j,k} \cdot s^i t^j u^k (1-s-t-u)^{n-i-j-k} = 0. \qquad (3)$$

Example 4.1 *Conversion from Power to Bernstein*

Consider, as a simple example, a quadric surface which Hermite interpolates a line $LN : (1-t, t, 0)$ with a normal $(0, 0, 1)$. The Hermite interpolation algorithm returns a 5 parameter family $f(x, y, z) = 0$ of algebraic surfaces, as in (2) with $n = 2$, where $c_{200} = \tau_1$, $c_{110} = 2\tau_1$, $c_{101} = \tau_4$, $c_{100} = -2\tau_1$, $c_{020} = \tau_1$, $c_{011} = \tau_5$, $c_{010} = -2\tau_1$, $c_{002} = \tau_3$, $c_{001} = \tau_2$,and $c_{000} = \tau_1$. For a given tetrahedron with vertices $P_{n00} = (2, 0, 0)$, $P_{0n0} = (0, 2, 0)$, $P_{00n} = (0, 0, 2)$, and $P_{000} = (0, 0, 0)$, the surface $f(x, y, z) = 0$ is transformed to $F(s, t, u) = 0$, as in (3) with $n = 2$, where $w_{000} = \tau_1$, $w_{001} = \tau_1 + \tau_2$, $w_{002} = \tau_1 + 2\tau_2 + 4\tau_3$, $w_{010} = -\tau_1$, $w_{011} = -\tau_1 + \tau_2 + 2\tau_5$, $w_{020} = \tau_1$, $w_{100} = -\tau_1$, $w_{101} = -\tau_1 + \tau_2 + 2\tau_4$, $w_{110} = \tau_1$, and $w_{200} = \tau_1$. □

Since the weights w_{ijk} of $F(s, t, u) = 0$ for a q-parameter family of algebraic surfaces have only q degrees of freedom, they can't be selected or modified independently. For example, suppose $w_1 = \tau_1 + \tau_2 + \tau_3 + 2\tau_4 - 1$, $w_2 = \tau_1 + \tau_2 + \tau_4 + 5$, and $w_3 = \tau_3 + \tau_4$. From these, we can derive the linear relation $w_1 - w_2 - w_3 - 6 = 0$ between the weights, and then an invariant $\Delta w_1 - \Delta w_2 - \Delta w_3 = 0$ which must be satisfied each time some of the weights are modified. (For notational simplicity, we assume the weights are indexed by

a single number instead of a triple.)

In general, using Gaussian elimination, we can derive a system of invariant equations

$$\begin{aligned} I_1(\Delta w_1, \Delta w_2, \cdots, \Delta w_c) &= 0 \\ I_2(\Delta w_1, \Delta w_2, \cdots, \Delta w_c) &= 0 \\ &\vdots \\ I_l(\Delta w_1, \Delta w_2, \cdots, \Delta w_c) &= 0 \end{aligned}$$

from the linear expressions of the weights

$$\begin{aligned} w_1(\tau_1, \tau_2, \cdots, \tau_p) &= w_1 \\ w_2(\tau_1, \tau_2, \cdots, \tau_p) &= w_2 \\ &\vdots \\ w_c(\tau_1, \tau_2, \cdots, \tau_p) &= w_c. \end{aligned}$$

Changing the weights can now be considered as moving from a weight vector $W = (w_1, w_2, \cdots, w_c)$ to another $W' = (w_1', w_2', \cdots, w_c')$, with the constraint that $\Delta W = W' - W$ is a solution of the system of invariant equations.

Example 4.2 *Shape Control of a Family of Quadric Surfaces*

The invariant system for the family of algebraic surfaces in Example 4.1 is $\Delta w_{010} + \Delta w_{000} = 0$, $\Delta w_{020} - \Delta w_{000} = 0$, $\Delta w_{100} + \Delta w_{000} = 0$, $\Delta w_{110} - \Delta w_{000} = 0$, $\Delta w_{200} - \Delta w_{000} = 0$. Figure 8 (left) shows an instance from the family where $w_{000} = -4$, $w_{001} = 4$, $w_{002} = 8$, $w_{010} = 4$,

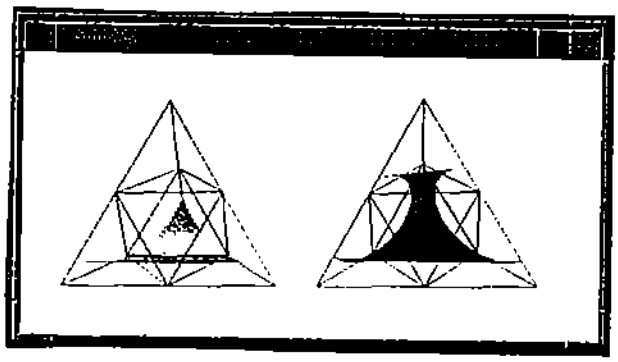

Figure 8: Shape control of a family of quadric surfaces

$w_{011} = 14$, $w_{020} = -4$, $w_{100} = 4$, $w_{101} = 12$, $w_{110} = -4$, and $w_{200} = -4$. Other Δw_{ijk} can be arbitrarily chosen as long as they satisfy the equations in the invariant system. □

Example 4.3 *Shape Control of a Family of Quartic Surfaces*

Figure 9 illustrates three different instances of a quartic family that smoothly joins three truncated orthogonal circular cylinders $CYL_1 : x^2 + y^2 - 1 = 0$ for $z >= 2$, $CYL_2 : y^2 + z^2 - 1 = 0$ for $x >= 2$, $CYL_3 : z^2 + x^2 - 1 = 0$ for $y >= 2$, corresponding to the three different values of $w_{000} = 0.35$, 0.6, and 0 for $P_{000} = (0, 0, 0)$. Figure 9 (left) shows the initial surface with weights determined by least squares approximation with a unit sphere centered at (1,1,1). Figure 9 (middle) shows the surface after increasing w_{000} to 0.6. As a weight w_{000} increases from a negative value, the surface approaches to P_{000}. The surface passes through P_{000} when $w_{000} = 0$ and gets separated into three irreducible components as w_{000} becomes positive, Figure 9 (right). □

Sometimes, we may want to see how the shape of a surface changes as a specific weight is modified. However, if a weight, say, w_1 is modified, then this modification affects other

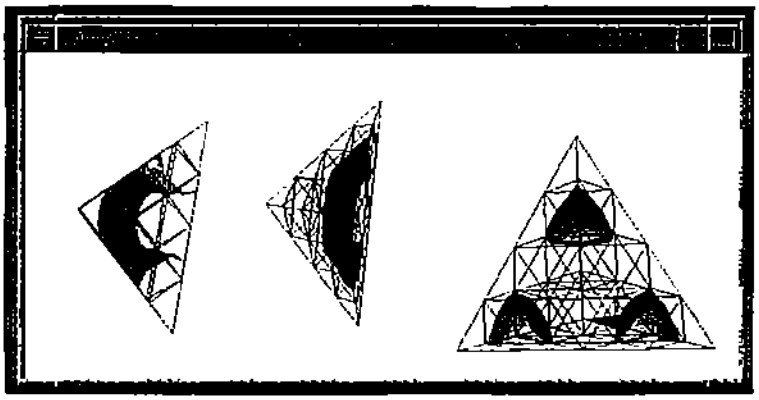

Figure 9: Shape control a family of quartic surfaces

weights as related in the invariant system. Usually, the linear system of invariant equations is underdetermined, yielding an infinite number of choices of Δw_i $(i = 2, 3, \cdots, c)$. Then, how can we select the other weights such that their effects to w_1 are minimized?

One possible heuristic is to minimize the 2-norm of $(\Delta w_2, \cdots, \Delta w_c)$, and hence the 2-norm $\|\Delta W\|_2 = (\Delta w_1^2 + \Delta w_2^2 + \cdots + \Delta w_c^2)^{\frac{1}{2}}$ of ΔW. For $\Delta w_1 = d$, we know that the linear system

$$\begin{aligned} I_1(d, \Delta w_2, \cdots, \Delta w_c) &= 0 \\ I_2(d, \Delta w_2, \cdots, \Delta w_c) &= 0 \\ &\vdots \\ I_l(d, \Delta w_2, \cdots, \Delta w_c) &= 0 \end{aligned}$$

has a solution $\Delta W^0 = (d, \Delta w_2^0, \cdots, \Delta w_c^0)$ where Δw_i^0's are expressed linearly through another set of free parameters $p_1, p_2, \cdots, p_s$. Hence, $\|\Delta W^0\|_2^2$ is a quadratic function $Q(p_1, p_2, \cdots, p_s)$ of the new parameters.

Since Q is quadratic, $Q(p_1, p_2, \cdots, p_s)$ is minimized at the solution of the linear system $\nabla Q(p_1, p_2, \cdots, p_s) = \mathbf{0}$. If the minimum of Q occurs at a point $(p_1^0, p_2^0, \cdots, p_s^0)$, then $\Delta W^0 = (d, \Delta w_2^0, \cdots, \Delta w_c^0)$ corresponding to the point defines the desired change of weights $w_2, \cdots, w_c$ having the minimum effect, in the least squares sense, on the shape of the surface. The instance surface corresponding to the new weights $W' = W + \Delta W^0$ will then reflect predominantly the effect of the change of w_1 by $\Delta w_1 = d$.

Example 4.4 *Heuristic Approach to Shape Control Using 2-Norm*

Consider the surface in Example 4.2 again. This time we wish to pull the patch more toward P_{002} (the topmost vertex in the figure), and hence set $\Delta w_{002} = -15$. From the invariant system in which Δw_{002} is replaced by -15, $\Delta w_{000} = \Delta w_{020} = \Delta w_{110} = \Delta w_{200} = p_1$, $\Delta w_{010} = \Delta w_{100} = -p_1$, $\Delta w_{001} = p_2$, $\Delta w_{101} = p_3$, $\Delta w_{011} = p_4$, and we obtain the quadratic function $Q(p_1, p_2, p_3, p_4) = 225 + 6p_1^2 + p_2^2 + p_3^2 + p_4^2$. Q has the global minimum at $p_1 = p_2 = p_3 = p_4 = 0$. Hence, the influence of the change of all the weights other than w_{002}, is minimized by setting to zero their Δw, that is, not changing them at all. This new instance is shown in Figure 8 (right). □

5 Conclusion and Future Works

SplineX is a distributed application system that manipulates different geometric patches in Bernstein-Bezier basis. It can be used to manipulate not only implicit but also parametric surface patches. As a component of Shastra, it communicates with other parts of Shastra, as a server or a client. We have exhibited this with Ganith and SplineX inter-operations.

Now SplineX provides the manipulation of 2D and 3D objects. One of our future developments is to extend it to 4D objects or higher. The extension will be applicable

to scientific visualization. And it is also challenging to extend the distributed rendering scheme of both implicit and parametric surface to 4D.

Also of theoretical interest, an open problem is the following. Given two planar subdivisions (or, specially, triangulations), is there an algorithm which merges them in linear time or a time bound less than $O(n \log n)$, such that the vertex set of the new subdivision (or triangulation) is the union of those of the two old subvisions (or triangulations)?

References

[ABB+91] V. Anupam, C. Bajaj, A. Burnett, M. Fileds, A. Royappa, and D. Schikore. XS: A Hareware Independent Graphics and Windows Library. Technical Report CSD-TR-91-062, Computer Sciences Department, Purdue University, August 1991.

[ABI+91] V. Anupam, C. Bajaj, I. Ihm, T. Dey, and I. Ihm. The SHILP Solid Modeling and Display Toolkit. Technical Report CAPO-91-29, Computer Sciences Department, Purdue University, 1991.

[ABR91] V. Anupam, C. Bajaj, and A. Royappa. The SHASTRA Distributed and Collaborative Geometric Design Environment. Technical Report CSD-TR-91-075, Computer Sciences Department, Purdue University, October 1991.

[BBB] R. Bartels, R. Beatty, and B. Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling.* Morgan Kaufmann Publisher, Inc.

[BBF91] B. Bailey, C. Bajaj, and M. Fields. The VAIDAK Medical Imaging and Model Reconstuction Toolkit. Technical Report CAPO-91-31, Computer Sciences Department, Purdue University, 1991.

[BC92] C. Bajaj and S. Cutchin. The GATI Client/Server Animation Toolkit. Technical Report CSD-TR-92-096, Computer Sciences Department, Purdue University, December 1992.

[BI92] C. Bajaj and I. Ihm. Algebraic surface design with Hermite interpolation. *ACM Transactions on Graphics*, 11(1):61–91, January 1992.

[BIW91] C. Bajaj, I. Ihm, and J. Warren. Higher order interpolation and least squares approximation using implicit algebraic surfaces. Technical Report CSD-TR-91-035, Computer Sciences Department, Purdue University, April 1991.

[BOS92] C. Bajaj, K. Okamura, and D. Schikore. The BHAUTIK Physical Analysis Toolkit. Technical report, Computer Sciences Department, Purdue University, 1992.

[BR91] C. Bajaj and A. Royappa. The ganith algebraic geometry toolkit. Technical Report CSD-TR-91-065, Computer Sciences Department, Purdue University, August 1991.

[Col86] Richard Cole. Searching and Storing Similar Lists. *Journal of Algorithms*, 7:202–220, 1986.

[dB87] C. de Boor. B-Form Basics. In G. Farin, editor, *Geometric Modeling*. SIAM, 1987.

[EGS86] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal of Computing*, 15:317–340, 1986.

[Far86] Gerald E. Farin. Triangular Bernstein-Bézier Patches. *Computer Aided Geometric Design*, 3(2), 1986.

[Far90] Gerald E. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, 1990.

[Sed85] T. W. Sederberg. Piecewise algebraic surface patches. *Computer Aided Geometric Design*, 2(1-3):53–59, 1985.

[SG86] Robert W. Scheifler and Jim Gettys. The X Window System. *ACM Transactions on Graphics*, 5(2):79–109, April 1986.

[ST86] Neil Sarnak and Robert E. Tarjan. Planar Point Location Using Persistent Search Trees. *Communication of the ACM*, 29(7):669–679, July 1986.