NURBS surfaces: a geometric primitive for cooperative visualization

H. Sánchez *1*, A. Moreno *2*, A. García-Alonso *3*, D. Oyarzun *2*

(1) Universidad de Extremadura
Centro Universitario de Mérida
Santa Teresa de Jornet , 38
06800 Mérida (Spain)

Téléphone : (+34) 924 387068 ext 2530
Fax : (+34) 924 303782
E-mail : sasah@unex.es

(2) VICOMTech
Mikeletegi Pasealekua, 57
Parque Tecnológico
20009 Saint Sebastián (Spain)

Téléphone : (+34) 943 30 92 30
Fax : (+34) 943 30 93 93
E-mail : {amoreno, doyarzun @vicomtech.es

(3) Euskal Herriko Unibertsitatea
Facultad de Informática
Manuel de Lardizabal , 1
28018 San Sebastián (Spain)

Téléphone : (+34) 943 015104
Fax : (+34) 943 219306
E-mail : agalonso@si.ehu.es

Abstract: The problem of visualizing graphical models described by NURBS surfaces at interactive frame rates is of great interest for the CAD/CAM industry, since many of their models are designed using this surface type. As current graphics hardware is optimised for rendering triangles, most efficient methods of rendering NURBS surfaces approximate them by polygonal representations. In this work we propose a predictive fixed-frame rate rendering of NURBS surfaces that controls the interactivity of their visualization and manipulation.

Key words: NURBS surfaces, fixed-frame rate tessellation, cooperative visualization.

1- Motivation

Collaborative visualization provides real time visual exploration and multiple-user interactions of three-dimensional models among geographically remote users. Per example, 3DSHARED [1][2] allows engineers, modellers and customers the virtual design and maintenance of engineering products through its three-dimensional replica.

In collaborative applications, the speed in the making of the following operations is vitally important: the transmission of the entire model, the transmission of messages that update the visualized scene and the generation of the updated final image. Improving these operations makes possible a real-time interaction with the three-dimensional model. At the moment, 3DSHARED supports three-dimensional models described by polygonal meshes. As the density of polygons in meshes is pre-calculated, it is possible to find difficulties in model transmission. Furthermore, large CAD models can stretch the capability of some hosts whose graphical features are limited. This fact complicates keeping a similar frame rate in all hosts that take part in a collaborative session.

A fundamental geometric entity in Computer Aided Design (CAD) is the Non-Uniform Ration B-Spline (NURBS) surface. This representation allows defining exactly both algebraic geometries and abstract surfaces standardizing the representation by a single mathematical equation. Moreover, thanks to their growing use in CAD/CAM, NURBS surfaces appear in the main neutral file formats for interchanging geometric data, like IGES and STEP. Pielg and Tiller [3] have studied NURBS surfaces in depth.

In Internet oriented applications, because of its implicit compression, NURBS surfaces are more convenient than polygonal meshes because the amount of information to be transferred among hosts is reduced. With regard to interactive visualization, its description allows to select the optimal level of detail in runtime. Thus, it is possible to manage the quality of the generated image as the function of, either the degree of interactivity to achieve, or the hardware features of a particular host. This means that, in different hosts participating in a collaborative session over the same

3D model, its polygonal approximation can be made up of a different number of polygons.

2- Related Work

2.1 – Rendering of NURBS surfaces

Because of its importance in computer graphics applications, the rendering of NURBS surfaces has been researched intensively in last three decades. In photo-realistic rendering algorithms, the colour intensity value of each screen pixel is generated directly from the mathematical description of the surfaces. These algorithms can be classified in based on: scan-line [4][5][6], ray tracing [7][8] and iso-curve sequences [9][10]. In hardware rendering [11][12][13], the graphical hardware is extended with more complex geometric primitives than the polygon. This hardware has VLSI architectures that make possible the computation of points and normal vectors of a surface. In approximation based rendering algorithms, more simple primitives like polygons [14][15][16] or points [17] approximate surfaces. These algorithms are generally much faster due to recent advances in graphics hardware.

In dynamic tessellation algorithms, meshes are generated in the interactive stage of the rendering pipeline. As they have to run as fast as possible, the tessellations they produce usually are of lower quality than those generated in preprocessing stage. However, there are two interesting facts. In first place, storing NURBS surfaces instead of storing polygonal representations saves a large percentage of available memory. This is especially interesting when using large models where many surfaces are not visible and their incorporation to the potentially visible surfaces set will be gradual. In second place, an appropriate tessellation can be obtained in runtime as the function of actual viewing conditions, the wanted interactivity degree or available graphical hardware features in a local computer. So, this kind of algorithms is very advisable for interactive visualization.

In dynamic tessellation, uniform decomposition is generally used because they are less time consuming than adaptive decomposition. Uniform decomposition tessellates the surface using a regular grid defined in the parametric domain. This does not guarantee that the resulting tessellation will be uniform. Anyway, it is possible to determine a parametric grid size that may produce polygons that meet certain restrictions. For instance, polygons that projected onto the screen will be within specific size bounds [14][16].

2.2 – Predictive fixed-frame rate LOD selection

After transforming surfaces into polygonal representations, most existing approaches for rendering interactively NURBS surfaces build static levels of detail upon them.

A predictive fixed-frame-rate LOD selection algorithm estimates the complexity of the frame to be rendered and selects appropriated levels of detail to ensure that the update deadline is never exceeded.

Funkhouser and Sequin [18] present a predictive LOD selection algorithm that adapts the discrete level of detail of visible objects to satisfy a constant frame rate. They used a cost/benefit paradigm that attempted to optimize the perceptual benefit of a frame against the computational cost. Mason and Blake [19] describe a hybrid of Funkhouser and Sequin's predictive LOD selection algorithm and imposters technique. Gobbetti and Bouvier [20]'s predictive LOD selection algorithm works with continuous LOD models. Zach et al. [21] presents a predictive LOD selection algorithm that incorporates discrete and continuous representations of each object.

3- Outline of the algorithm

Given the set of NURBS surfaces that completes a model, the predictive fixed-frame rate tessellation algorithm can be divided into a pre-processing stage and an interactive rendering stage.

In the pre-processing stage the following actions are done:

- Reading NURBS surfaces
- Computation of geometric data (see Section 6)

The interactive rendering stage consist of the following two main operations:

- Culling non visible surfaces
- Predictive-fixed frame rate tessellation of visible surfaces

In interactive visualization, the speed of interaction with the 3D model (quantified by the frame rate) has priority over the image quality. One form of controlling the frame rate consists on explicitly bounding the number of polygons that forms tessellations of visible surfaces while maintaining as much quality as possible. As it will be seen later on, uniform tessellation permits the implementation of a fixed frame rate predictive algorithm.

4- Visibility Control

We implement two different culling techniques at surface level: view frustum and back-face culling.

The view frustum culling works by testing the bounding sphere of each surface against the current view-frustum, which is defined by the four sides of a truncated pyramid.

The back-face culling algorithm is based on the clustered back-face culling used by Zhang and Hoff III[22] for polygonal models. We extend that idea using a discretization of the *normal surface* of each NURBS surface. Unitary normal space is partitioned in small pyramids named clusters. The apex of all pyramids is the centre of a unitary cube. Each face of the cube is subdivided in regular cells. The base of each pyramid is one of the cells. In pre-process, we determine for each surface which clusters are intersected by its normal vectors. In each frame, front-facing clusters are determined in constant time based on the representative normal vectors of each cluster (normals of its corners). If one cluster is front facing, surfaces whose normal vectors intersect this cluster are deemed visible.

5- Tessellation criterion

We propose a new tessellation criterion based on the projection onto screen space of surface bounding sphere. As we will see, we extended that method using surface geometric information. With our tessellation criterion, triangle edge length projected onto screen space can be bounded.

In the pre-process phase we compute the maximum lengths of iso-edges of the control polygon in each parametric direction *(lu_{max}; lv_{max})*:

$$lu_{\max} = \max_i \sum_{j=0}^{m-1} \left\| P_{i,j} - P_{i,j+1} \right\|$$

$$lv_{\max} = \max_j \sum_{i=0}^{n-1} \left\| P_{i,j} - P_{i+1,j} \right\|$$

where $P_{i,j}$ $((n+1)\times(m+1))$ are the control points that define a NURBS surface. We also compute the aspect ratio among these lengths $ar = lu_{max}/lv_{max}$.

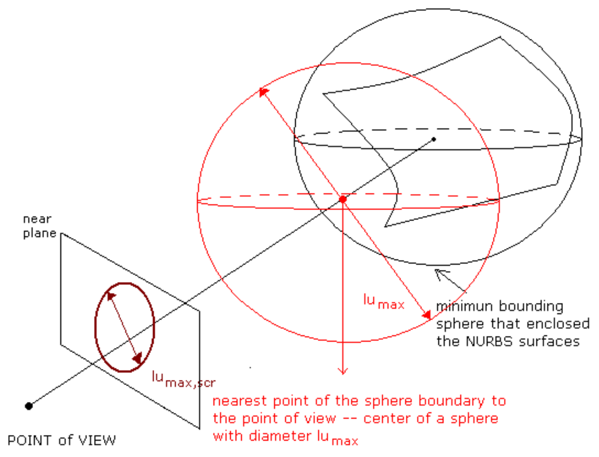
In the interactive rendering phase we use this data to determine the constant step size that will be used in the parametric space to generate the vertices for the required tessellation. We define an sphere of lu_{max} diameter, with its centre placed in the nearest point to the camera view point of the surface bounding sphere (see Figure 2). Then, step sizes in each parametric direction (n_u, n_v) are:

$$n_u = \frac{u_{\max} - u_{\min}}{\dfrac{lu_{\max,scr}}{\lambda} - 1}$$

$$n_v = \frac{v_{\max} - v_{\min}}{\dfrac{lu_{\max,scr}}{\lambda \cdot ar} - 1}$$

where (u_{max}, u_{min}) and (v_{max}, v_{min}) are the dimensions of the parametric domain in each parametric direction, $lu_{max,scr}$ is the screen space projection of lu_{max}, and λ, the triangle edge length bound parameter, whose use and meaning will be explained immediately.

A tessellation of a surface is determined by its $lu_{max,scr}$ and λ. $lu_{max,scr}$ is the function of: lu_{max}, the centre of the surface bounding sphere, the position of the point of view and the position of the projection plane. For a constant value of λ, the number of polygons of a tessellation will be reduced or increased if the distance among the point of view and the surface increases ($lu_{max,scr}$ smaller) or decreases ($lu_{max,scr}$ larger).



near plane
lu_{max}
minimun bounding sphere that enclosed the NURBS surfaces
$lu_{max,scr}$
nearest point of the sphere boundary to the point of view -- center of a sphere with diameter lu_{max}
POINT of VIEW

Fig. 2: Tessellation criterion.

The value of λ is used as a global quality bound that is applied to all the surfaces in the scene. The effect of changing λ maintaining constant the rest of parameters implies: with increases of λ, step sizes decrease and, therefore, less quality tessellations are generated; inversely, reducing λ involves more quality tessellations. Consequently, if the system detects that the frame rate is lowering, or the memory cost is excessive, the logical action will be increasing the value of λ.

From a quality point of view, a new tessellation of a surface is made only when the following happens:

$$\left| lu_{\max,scr,current} - lu_{\max,scr,required} \right| > K \cdot lu_{\max,scr,current}$$

where K > 0 controls whether a new tessellation must be built.

The behaviour of K is simple and intuitive. If K is assigned a large value, the quality difference between two consecutive tessellations is also large, and switching between them will produce the popping effect. On the other hand, if K is small, it is possible that the quality increase of the new tessellation will not compensate the effort of its generation.

Independently of the value of *K*, the frequency of tessellations decreases as the distance between the surface and the point of view decreases. Equally, if the value of *K* increases, the tessellations are more frequent.

6- Computational Cost of Tessellation and Rendering

If the value of λ is constant during the visualization, the quality of tessellations increases as the distance between the model and the point of view decreases. Consequently it is possible that the amount of polygons exceeds the memory or the rendering capability of the graphics hardware.

The total computational cost of generating and rendering the tessellation of potentially visible surfaces is approximated by:

$$C_{total} = \sum_{i=1}^{N} C(S_i)$$

where N is the set of surfaces and $C(Si)$ is the cost of generating and rendering the tessellation of a particular surface, Si. This time is estimated as:

$$C(S_i) = C_{gen}(S_i) + C_{rend}(S_i)$$

The time required to generate a tessellation of a surface depends on its number of vertices and the time required to evaluate a vertex (a vertex is composed of the three-dimensional surface point and its associated normal vector): given $lu_{max,scr}$, ar and λ, the temporal cost of generating the tessellation of the surface S_i is:

$$C_{gen}(S_i) = \left(\frac{lu_{\max,scr}(S_i)}{\lambda} - 1 \right) \cdot \left(\frac{lu_{\max,scr}(S_i)}{\lambda \cdot ar(S_i)} - 1 \right) \cdot t_{eval}$$

where t_{eval} is the required time to evaluate a vertex with a particular surface evaluation method.

Respect the rendering of surfaces, Funkhouser and Sequin [18] predict the temporal cost of the *Geometry* stage for a polygonal object as a lineal combination of the number or polygons and vertices that compose the object weighted by coefficients that depends on the hardware features and the rendering algorithm (wireframe, Gouraud, Phong,...). For the *Rasterizer* stage, the temporal cost is proportional to the number of pixels covered by the screen projection of the object.

Calculating the number of polygons and vertices is straightforward. However, determining the number of pixels that covers a projected surface is computationally expensive. It requires calculating silhouette curves and their projection to screen space in runtime. This value can be approximated projecting and calculating the projected area of a bounding volume of the surface. Another possibility is pre-compute the area for a set of projections and in the interactive phase obtain the number of pixels as the function of the distance and those pre-computed areas.

7- Predictive fixed-frame rate tessellation

We propose a predictive fixed-frame rate tessellation algorithm based on the parameters exposed in previous sections.

We define $\boldsymbol{tr_{max}}$ as the time available for the generation of one frame. The user or the application sets the value of this parameter (e.g. 50 msec.). It should be noted that in each frame all the tessellations are rendered to create one image, but not all visible surfaces must be tessellated for that frame, only a few require a new tessellation (see section 5). So, tr_{max} must be distributed between those two tasks, plus the culling phase. However, sometimes might happen that there is not enough time within a frame to render the image and to create all the new tessellations needed. The algorithm predicts the time required to perform the different tasks and gives priority to the generation of a new image per frame. However, this is not an absolute policy, if required, it reserves a minimum time fraction per frame for the generation of tessellations that will reduce the display cost. This policy should lead to less complex tessellations, reduce the time required for the generation of one frame and a new equilibrium should be achieved. In first place we define some variables then we propose the algorithm:

- $\boldsymbol{tr_v}$ is the consumed time for the culling operation within a given frame, so the time left for tessellate and render in that frame is : $\boldsymbol{tr_{max} - tr_v}$
- $\boldsymbol{te_f}$ is the estimated time for the generation of one frame. This time is obtained summing the estimated tessellation and rendering times for that frame: $\boldsymbol{te_f = te_t + te_r}$. Making this estimation is the first task performed in each frame once the culling operation is done
- $\boldsymbol{tr_t}$ is the time that is really spent within a frame to generate tessellations and render them

With respect to parameter λ we use the following notation:

- $\boldsymbol{\lambda_{obj}}$ is the objective quality for all the tessellations. As we will see, it is an objective that sometimes requires several frames before it is achieved, so it may happen, when rendering one frame, that existing tessellations have a different $\boldsymbol{\lambda}$ value.
- $\boldsymbol{\lambda_\Delta}$ is the increment that is applied to λ_{obj} when it is necessary to increase or reduce the quality
- $\boldsymbol{\lambda_{Si}}$ is the current quality associated to the tessellation of surface S_i
- $\boldsymbol{\lambda_{max}}$ is the current largest λ_{Si}. Its possible values are λ_{obj} or $\lambda_{obj} + \lambda_\Delta$ (note that λ_Δ can be +/-)

Rendering a frame consists of the following operations:

- Determine which surfaces require a new tessellation
- Predict the computational cost of tessellating and rendering the required surfaces, te_f.
- Determine the time consumed by these two tasks, tr_v

If the time available to generate the frame is larger than the estimated time ($te_f < tr_{max} - tr_v$), the following operations will be done (render + increase quality):

- Tessellate the surfaces that require a new tessellation.

- Render.
- If the up to now consumed time is less than tr_{max}, surfaces with $\lambda_{Si} > \lambda_{obj}$ are tessellated again with more quality ($\lambda_{Si} = \lambda_{obj}$). If the available time is over, this process is interrupted. If the quality of all surfaces is λ_{obj}, this value is decremented in λ_{Δ}: the objective quality is increased.

Otherwise, if the available time to generate the frame is smaller than the estimated time ($te_f < tr_{max} - tr_v$), the following operations will be done (render + increase quality + allow for lower frame rates):

- If $\lambda_{max} = \lambda_{obj}$, the objective quality is reduced in λ_{Δ}.
- Decide how much time will be dedicated to generate coarser tessellations, tr_t. This election is made as function of the estimated time of rendering, te_r. The following simple scheme is suggested. This election is not obvious: a iterative-predictive loop that estimates what surfaces should be tessellated to maximize the rendering time saving could be use.

$$te_r > tr_{\max} : tr_t = tr_{\max} \cdot 0{,}5$$
$$te_r \le tr_{\max} : tr_t = tr_{\max} - te_r \cdot 0{,}5$$

- During this time, tr_t, surfaces with $\lambda_{Si} < \lambda_{obj}$ are tessellated again reducing their quality ($\lambda_{Si} = \lambda_{obj}$). If the available time is over, this process is interrupted and the rendering phase begins. This simplification will continue in the next frame.
- If the previous operation has been completed and the available time is not over, surfaces that require a new tessellation will be tessellated again.
- Render.

8- Description of 3DSHARED

3DSHARED is composed of two modules: *3D API* and *Collaborative API*. The 3D API takes care of the aspects related with the visualization of three-dimensional models. The Collaborative API ensures the correct communication among the members of the cooperative session. As the design and implementation of both modules is independent, 3DSHARED offers the functionalities of both modules in the same interface. Consequently, the cooperative visualization of three-dimensional models is allowed.

8.1 – Dynamic client/server architecture

The Collaborative API represents a new communication architecture based on two well-known architectures: client/server and distributed [23].

In the client/server architecture, the server manages the transfer of messages among the participants. It receives messages from clients and replies them to the other hosts. This is a centralized management. Although this architecture is easily implemented, a server must firstly be started in order to allow the connection of clients [24]. Even more, a failure in the server causes the whole system failure of the system and this kind of architecture is poorly scalable. These problems can be solved with the hybrid client/server architecture [25]. However, these solutions need to develop and support two different applications (the server and the client).

In the distributed architecture, all the hosts develop the same function. Therefore, only one application exists. As there is no server role, it is necessary to hold connections among all the hosts [26]. However, in our implementation, most of these connections are useless during the session because each host will only use the connections with the host that owns the control of the system. Therefore, the application looses the management simplicity.

We have designed a hybrid architecture, which avoids these problems: the *dynamic client/server architecture*. It is based on a "peer to peer" strategy [27] and any node can take the server role. At any time a client host can ask for the control of the system and shift from client role to server role. The server node, also called controller, is the manager of the communications. If the server node falls down there is a replacement mechanism that assigns the server role automatically to a client host.

Depending on the situation of the collaborative session the Collaborative API acts either as client or server, in a way that is transparent to the user. This avoids the development and co-existence of two different modules. If the system control is transferred to another host, the old controller host switches the role with the new controller (see Figure 3).

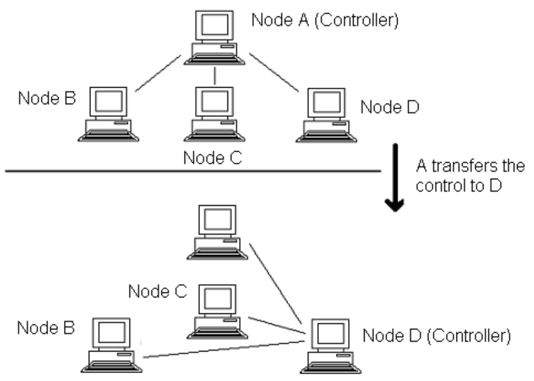


Fig. 3: Dynamic client/server architecture.

When a participant wants join to the session, it can connect to any participant. If this host is not the controller, the application intelligently redirects the connection to the controller. A user can be accepted, rejected or asked to join a cooperative session. This allows a robust management of the group.

The access control to the cooperative session is managed by the token mechanism that establishes that only the controller can interact with the model. Changing the control over the system can be accomplished for two reasons: the controller does not want to own the control and gives it to a client host; a client host wants to become the controller and requests it. This provides a higher flexibility to the system: the capability of transferring the token dynamically during a cooperative session.

8.2 – Communication of interactions

In a collaborative session, the controller host must communicate the interactions its user is performing over the model. Following these commands the participant hosts can generate the same scene on their respective screens.

3DSHARED uses a short message policy. These messages are sent only when there is a change in the state of the scene (position, orientation or colour change, etc.). As a consequence, the information flow among the participants is low and constant, achieving in most cases nearly real-time synchronization.

Using TCP/IP protocol, secure and ordered transmission is provided. Therefore, it is not necessary to use specific hardware like multicast routers or D class IP addresses or other solutions like JavaGroups [28] or Ensemble [29].

The possibility of sending text messages has been integrated in the application. This functionality (chat) is very useful in collaborative applications.

The user of the controller node is able to manipulate the user interface in order to change the visualization parameters. The 3D API captures these interactions. Each controller's action over the interface generates an update message that is queued in order to be sent to the clients.

We have been very concerned about the lack of bandwidth for the communications. Thus we have built an option to control the data transfer flow: the controller can uphold the transfer of messages. So clients will not receive changes in the scene as they occur. Changes are queued and they are sent only when the user of the controller node requests the application to send them. Consequently, the 3D API provides two communication types: continuous mode and send by order mode. Viewing the object in motion is an important cue to help our mind in the recognition of 3-D models; for this reason continuous mode is the most used. However, if the Internet is very busy, then it is worthwhile to lose this cue to avoid the annoying effect of display lag.

The Collaborative API of the participant hosts receives the messages. These messages are interpreted. If a message announces the opening of a file, the system checks whether that file is locally stored or not. If it is not, its transfer is requested and the application is interrupted until transmission is finished. If a message encapsulates an update of the scene, it is queued and the 3D API interprets it and reflects the changes in the scene that encapsulates it. The communication process can be seen in the Figure 4.

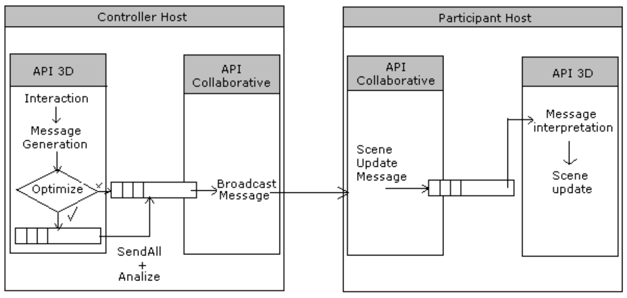


Fig. 4: Communication of the update messages between the 3D API and the Collaborative API.

The Collaborative API is a general purpose API [30]. It contains the client/server architecture and the communication protocol but not the message policy. This feature allows its implantation in several domains. It is only required to define the message policy and how the 3D API must act when it receives the message.

8.3 – User interface of 3DSHARED

Each user will have one instance of the same program, which is presented in Figure 5. It has three communication tools: a chat tool, a file transfer tool and a 3D-window where the same view for all the participants in the session appears.

The chat tool helps to check the connections. It is usually needed when the Internet is very busy. It also provides a simple mechanism to log the comments along the session. It can be used instead of the telephone or as a supplement to voice communication when a collaborative session is held between people of different languages, a usual problem in Europe. The list of connected nodes is available in the interface.

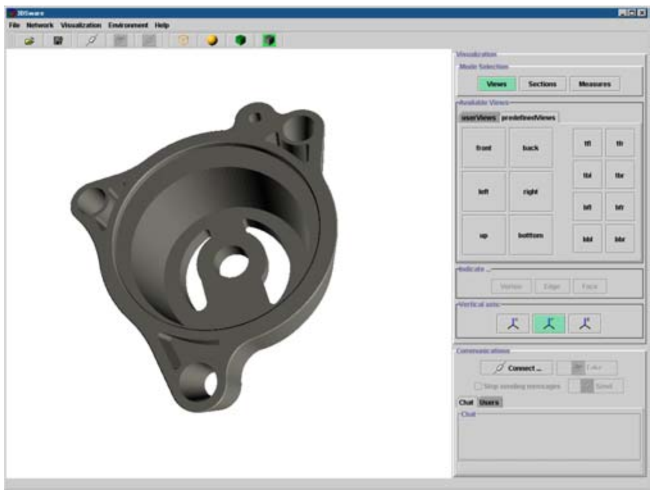


Fig. 4: Cooperative Visualization Tool: 3DSHARED.

The file transfer is required nearly in every session, so we have integrated a tool to make this task more comfortable.

The cooperative visualization window is used to: verify concurrently the design of the model and signal a particular feature to the other participants (sometimes to explain a doubt, other times to ask about a detail).

Now we will refer to some visualization features of the application. The controller can select any of the 14 predefined views, change the viewing conditions freely, or select one of the user views. The user can store or remove views from a list, which can be reused in several sessions. Switch buttons define which axe of the model is mapped vertically on the screen. Cutting planes that remove part of the model are used to see the inside and get a better understanding of the structure of the model. Verification may be done visually, but the caster usually needs to make some measurements to check angles, thickness, radiuses, distances, etc., while the discussion with the modeller is carried out in the CSCW session. All of these functionalities are sent from the controller to all clients.

9- Conclusions

In this work we propose a framework for the interactive visualization of models described by NURBS surfaces.

For the time being the following operations of the framework have been implemented:

- a visibility control composed by a extension of the Zhang and Hoff III's clustered backface culling and the view frustum culling,
- a uniform tessellation algorithm guided by a geometric tessellation criterion. It does not assume continuity conditions of surfaces,
- a tessellation policy that allows the user to configure the tessellation process dynamics.

Moreover, we propose a predictive fixed-frame rate scheme. It makes possible to adjust the quality of the meshes predicting the cost of generating and rendering tessellations. This control can be done thanks to the parameters we have defined to control the tessellation process.

As future work, the main operations to be done are:

- to extend the visibility control with hierarchical data structures and frame coherence.
- to implement benchmarks of the proposed frame rate control and explore new solutions
- to extend the proposed framework for trimmed NURBS surfaces.

Acknowledges

This work has been carried out with the help of a Researching Personal Training Grant from Basque Government to Héctor Sánchez and CICYT grant TIC99-0252. Héctor Sánchez also wish to thank Prof. Subodh Kumar and Jatin Chhuganni for the time spent discussing fast tessellation algorithms during his stay at Johns Hopkins Uni-versity.

10- Bibliography

[1] D. Borro, I. Recio, H. Sánchez, A. García-Alonso and L. Matey. CSCW for foundry design using Java3D. ACM 2000 Conference on Computer Supported Cooperative Work, Philadelphia, December 2000.

[2] D. Borro, I. Recio, C. Pedrinaci, H. Sánchez and A. García-Alonso. Peer-to-peer techniques applied to the Cooperative Visualization of CAD Models. Proceedings of MICAD 2003, Paris, pp. 8–13, April 2003.

[3] L.A. Piegl and W. Tiller. The NURBS Book. Monographs in Visual Communication. Springer, 2º edition, 1997.

[4] F. Blinn. A scanline algorithm for computer display of curves surfaces. Proceedings of the 5th annual conference on Computer graphics and interactive techniques, 1978

[5] J.T. Whitted A scanline algorithm for displaying parametrically defined surfaces. ACM Computer Graphics, vol. 12, nº. 3, pp 8-13, 1978

[6] J.M. Lane, L.C. Carpenter, J.T. Whitted and J.F. Blinn. Scan line methods for displaying parametrically defined surfaces Communications of ACM, vol. 23, nº. 1, pp. 23-34, 1980.

[7] K. Qin, M. Gong, Y. Guan and W. Wang, A New Method for Speeding Up Ray Tracing NURBS Surfaces", Computers & Graphics, vol. 21, nº. 5, pp. 577-586, September-October, 1997

[8] W. Martin, E. Cohen, R. Fish and P. Shirley. Practical Ray Tracing of Trimmed NURBS Surfaces. Journal of Graphics Tools vol. 5, nº. 1, pp. 27-52, 2000

[9] S. Chang, M. Shantz and R. Rocchetti. Rendering Cubic Curves and Surfaces with Integer Adaptive Forward Differencing. Computer Graphics (SIGGRAPH), vol. 23, nº 3, pp. 157-166, Jul. 1989

[10] G. Elber and E. Cohen. Adaptive isocurve-based rendering for freeform surfaces. ACM Transactions on Graphics, vol. 15, nº 3, pp. 249-263, 1996

[11] R. Bedichek, C. Ebeling, G. Winkenbach and A. D. DeRose. Rapid low-cost display of spline surfaces. Advanced Research in VLSI: Proceedings of the 1991 UCSC Conference, MIT Press, Cambridge MA, pp. 340-355, March 1991

[12] M. Gopi and S. Manohar. A Unified Architecture for the Computation of B-Spline Curves and Surfaces. IEEE Transactions on Parallel and Distributed Systems, vol. 8, nº. 12, pp. 1275-1287, 1997

[13] M. Boó, J. D. Bruguera and E. L. Zapata Parallel architecture for the computation of NURBS surfaces. Proceedings of SPIE: Media Processors 2000, pp. 37-48, 2000

[14] A. Rockwood, K. Heaton and T. Davis. Realtime rendering of trimmed surfaces. ACM Computer Graphics (SIGGRAPH Proceedings), vol. 23, nº. 3, 1989

[15] L.A. Piegl and W. Tiller. Geometry-based triangulation of trimmed NURBS surfaces. Computer-Aided Design, vol. 30, nº 1, pp. 11-18, 1998

[16] S. Kumar. Interactive Display of Parametric Spline Surfaces. PhD Thesis, University of North Carolina, 1996

[17] J. Chhuganni and S. Kumar. Budget Based Sampling of Parametric Surfaces. to appear in ACM 3D Interactive Graphics 2003.

[18] T.A. Funkhouser and C.H. Sequin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. Computer Graphics (SIGGRAPH 93 Proceedings), 27, pp. 247–256, August 1993.

[19] A. Mason and E.H. Blake. Automatic Hierarchical Level of Detail Optimization in Computer Animation. Computer Graphics Forum, 16(3), pp. 191–199, 1997.

[20] E. Gobbetti and E. Bouvier. Time-critical multiresolution scene rendering. IEEE Visualization, pp. 123–130, 1999.

[21] C. Zach, S. Mantler and K. Karner. Time-critical Rendering of Discrete and Continuous Levels of Detail. Eurographics Workshop on Rendering, pp. 1–8, 2002.

[22] H. Zhang and K.E. Hoff III. Fast Backface Culling Using Normal Mask. Symposium on Interactive 3D Graphics, pp. 103–106, 1997.

[23] R. Gossweiler, R.J. Laferriere, M.L. Keller, and R. Pausch. An Introductory Tutorial for Developing Multi-User Virtual Environments. Presence: Teleoperators and Virtual Environments vol. 3, nº. 4, pp. 255-264, 1994.

[24] D. Brutzman. Graphics Internetworking: Bottlenecks and Breakthroughs. In Digital Illusions. Dodsworth, C. (ed.), pp. 61-97, Addison Wesley, 1997.

[25] K. Saar. VIRTUS: A collaborative multi-user platform. Proceedings of the VRML'99 Symposium, pp. 141-152, Paderborn, Germany, 1999.

[26] M.R. Macedonia and M.J. Zyda. A Taxonomy for Networked Virtual Environments. IEEE Multimedia vol. 4, nº. 1, pp. 48-56, Jan.-Mar., 1997.

[27] D. Clark. Face-to-Face with Peer-to-Peer Networking. Computer, vol. 34, nº. 1, pp. 18-21, January 2001.

[28] B. Ban. JavaGroups – A Reliable Multicast Communication Toolkit for Java. Home Page: www.cs.cornell.edu/Info/Projects/JavaGroupsNew/index.html, 1999.

[29] T. Clark. Ensemble – Distributed Communication System.Home Page: www.cs.cornell.edu/Info/Projects/ensemble, 2002.

[30] C. Pedrinaci. Integración y validación de un sistema de visualización cooperativa en Internet. Available at http://scsx01.sc.ehu.es/ccwweb3d/docs/pfc_pedrinaci.doc.