NURBS in VRML

Holger Grahn, Thomas Volk, Hans J. Wolters*

blaxxun interactive
*Hewlett Packard Laboratories

ABSTRACT

In the days of VRML 1.0, NURBS seemed to be too complex to be adapted to the specification. Current development of hardware compels us to reevaluate this idea: While CPU clocks break the 1-GHz barrier, users still have to cope with 56K modems. NURBS meet exactly these demands. A NURBS description is a compact storage form, but its evaluation requires more computational effort. In addition, NURBS can be utilized for morphing effects and they provide a means for a smooth LOD. Adopting trimmed NURBS allows visualization of complex CAD models in VRML. This paper gives an overview of the proposed nodes and their implementation and applications. We take a closer look at LOD, animations and trimmed NURBS. Finally, we look ahead and briefly touch on an emerging new representation, subdivision surfaces.

Keywords: NURBS, trimmed NURBS, parametric surfaces, subdivision surfaces

1. INTRODUCTION

During the design of VRML 1.0, the specification was closely aligned with OpenInventor [sgi]. A lot of concepts of Inventor were adopted in VRML 1.0 and passed on to VRML 2.0. Nevertheless, the NURBS primitives of Inventor were left out, because NURBS had been too CPU-intensive at the time and the description was seen as too complex compared to conventional primitives. Nowadays the average Internet connection cannot feed the CPU and the modern graphics subsystems a sufficient amount of data to keep them busy. That means that the applications are bandwidth-limited. Hence, adopting NURBS to the VRML specification seems to be the next logical step in the further development of Web-based graphics when taking into account the current development of hardware and infrastructure.

Holger Grahn, holger.grahn@blaxxun.de, blaxxun interactive, Elsenheimerstr. 61-63, 80687 Munich, Germany.
Hans J. Wolters, wolters@hpl.hp.com, Hewlett Packard Laboratories, 1501 Page Mill Rd, Palo Alto, CA 94304
Thomas Volk, thomas.volk@blaxxun.de, blaxxun interactive, Elsenheimerstr. 61-63, 80687 Munich, Germany.

In the first section, we will give a short introduction to NURBS and trimmed NURBS. A comprehensive description of NURBS can be found in [PT95] and [Far96]. In the second section, the proposed nodes are introduced; the complete nodes proposal can be found at [bla1]. In the third section, we will address implementation issues. The results of our implementation are shown in the following section.

NURBS

A point on a NURBS surface is defined by:

$$\boldsymbol{Q(u,v)} = \frac{\sum_{i=0}^{m_u}\sum_{j=0}^{m_v} B_{i,k_u}(u) B_{j,k_v}(v) w_{i,j} V_{i,j}}{\sum_{i=0}^{m_u}\sum_{j=0}^{m_v} B_{i,k_u}(u) B_{j,k_v}(v) w_{i,j}}$$

u,v	parameters of the surface
B	basis functions
k	orders in u and v direction
V	mesh of control points
w	weights

The basis functions are defined as follows:

$$B_{i,1}(u) = \begin{cases} 1 & \boldsymbol{u_i \le u < u_{i+1}} \\ 0 & \boldsymbol{otherwise} \end{cases}$$

$$B_{i,r}(u) = \frac{u - u_i}{u_{i+r-1} - u_i} B_{i,r-1}(u) + \frac{u_{i+r} - u}{u_{i+r} - u_{i+1}} B_{i+1,r-1}(u)$$

$$U = \{u_0, \ldots, u_{m_u}\}$$

U is the knot vector containing a nondecreasing sequence of real numbers.

By stepping through the u and v domains and evaluating the equation for points on the surface, a grid of sample points can be produced. Triangle strips can be generated by stepping through the u domain at two fixed v values.

The normals are computed by taking the cross product of the surface derivatives

$$n = \frac{\partial}{\partial u} Q(u,v) \times \frac{\partial}{\partial v} Q(u,v)$$

and normalizing the resulting vector.

This evaluation scheme is referred to in the literature as uniform tessellation. This method is staightforward to implement, less CPU-intensive and suitable for parallel computing. For a fixed tessellation it is possible to precompute all the necessary basis functions B_i (u) and B_j(v). In addition, some properties of NURBS can be exploited. The control points are invariant to transformations. Thus the small number of control points can be transformed instead of the huge number of output vertices. The vertices are lighted in screen space afterwards. Furthermore, the convex hull property of NURBS states that the surface or curve lies completely within the convex hull formed by its control polygon. Hence the control polygon can be used as bounding box for culling. It is also known that by repeatedly performing subdividision via knot insertion [Far96] the control polygon converges quadratically to the surface [Dahmen86]. By exploiting this fact, we can compute very tight bounding hulls.

As a drawback of a fixed step size the surfaces can be oversampled or undersampled: a flat surface may be broken up into a very fine mesh, or a surface of high curvature may be represented by a coarse mesh. This problem is adressed in the adaptive subdivision scheme as described in [Pet94]. Adaptive tessellation approximates the surface more accurately, especially in cases of highly varying curvature, but is more CPU-intensive. In our approach, we use a uniform tessellation due to its lesser computational requirements. If the NURBS surface is parametrized appropriately, then the placement of the knot lines reflects the surface properties well. In areas of dense knot lines the surface will be more complex than in areas with sparse knotlines. Hence a tessellation formed by dividing knot intervals into a fixed number of subintervals will sample the surface accurately.

Boundary computation

There is considerable literature on step size computation for uniform tessellation such as [RHD89], [FMM86], [AES91], [KML96]. There are two categories of algorithms, the size criterion and the deviation criterion. The size criterion determines the bound based on the size of the resulting triangles in screen space. Applying this step size to uniform tessellation still means that smooth areas are oversampled because the step size is related to the maximum curvature. The deviation criterion computes a bound on the maximum deviation of the tessellated surface from the NURBS surface. The deviation criterion produces good results but is computationally expensive.

Cracks

Since adjacent surfaces need not necessarily be tessellated with the same step size, cracks will appear at the patch boundaries. If the boundary curve of two surfaces has an identical parametric representation, a strip of coving triangles can be generated at the boundary as described in [FMM86, RHD89]. If the control points of the boundary do not match, boundary curves have to be tested for intersection to find out if they are identical [KML95]. Especially when rendering trimmed NURBS, the trimming curves have to be checked. Patches may have different trimming curves in terms of control points representing the curve.

Trimmed NURBS

To describe arbitrary shapes, we have to introduce trimmed NURBS patches. Here, so-called trimming loops which are specified in parameter space of a surface mark invalid regions of the NURBS patch domain. Especially in the CAD domain, trimmed NURBS are used to design objects with fluid shapes like ship hulls and aircraft or car bodies. Also in solid modelling patches containing holes are represented in trimmed form. Trimming loops consist of one or more trimming curves, which are 2D NURBS curves or piecewise linear curves, lying in parameter space of the surface and forming a closed loop. Degeneracies like selfintersecting trimming loops and intersecting loops need special attention, in that they have to be split up into non intersecting loops.

Two different approaches of tessellating trimmed surfaces can be found in literature: In [Luk93] and [LC93] the B-Spline representation is used for rendering. The rendering algorithm involves the computation of intersections of trimming curves with the iso-lines and triangulation. Since these operations are simpler and faster to perform on Bezier-representations than on B-Splines, algorithms presented in [RHD89], [KML96] and [AES94] first convert NURBS surfaces to Bezier surfaces. The algorithm in [RHD89] partitions each trimming curve into monotonic segments followed by a special triangulation at the patch boundaries. We share this approach in our implementation which is stable and straightforward to implement. The monotonic subdivision and the triangulation may become a bottleneck [RHD89]. More recent results as [KML96] present more efficient algorithms for the triangulation, the computation of tight bounds and the exploitation of frame coherence.

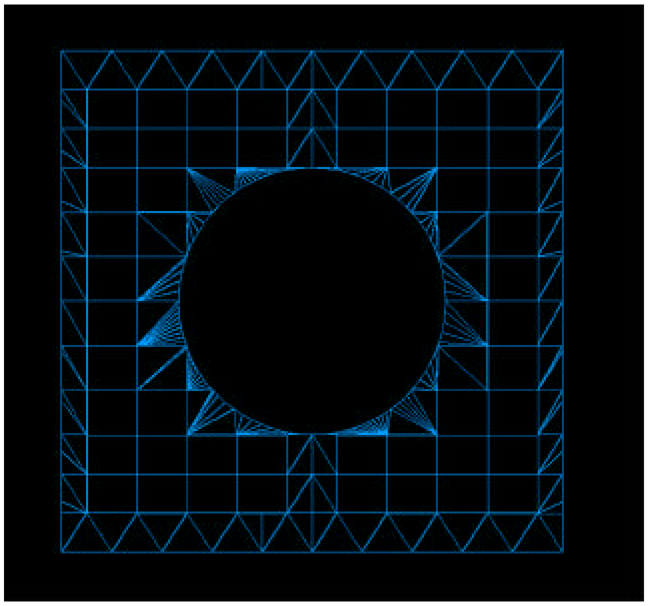

Figure 1: Trimmed NURBS patch rendered with the glu tessellator.

2. PROPOSED NODES

NurbsSurface

```
NurbsSurface  {
field         SFInt32 uDimension   0     # [0, ∞)
field         SFInt32 vDimension   0     # [0, ∞)
field         MFFloat uKnot        []    # (-∞,∞)
field         MFFloat vKnot        []    # (-∞,∞)
field         SFInt32 uOrder       3     # [2, ∞)
field         SFInt32 vOrder       3     # [2, ∞)
exposedField MFVec3f controlPoint  []    # (-∞,∞)
exposedField MFFloat weight        []    # (0, ∞)
exposedField SFInt32 uTessellation 0     # (-∞,∞)
exposedField SFInt32 vTessellation 0     # (-∞,∞)
exposedField SFNode  texCoord      []
field         SFBool  ccw          TRUE
field         SFBool  solid        TRUE
}
```

uDimension and ***vDimension*** define the number of control points in the u and v dimensions.

uOrder and ***vOrder*** define the order of surface. From a mathematical point of view, the surface is defined by polynomials of the degree *order-1*. The order of the curves uOrder and vOrder must be greater than or equal to 2. An implementation may limit uOrder and vOrder to a certain number. The most common orders are 3 (quadratic polynomial) and 4 (cubic polynomial), which are sufficient to achieve the desired curvature in most cases. The number of control points must be at least equal to the order of the curve. The order defines the number of adjacent control points that influence a given control point.

controlPoint defines a set of control points of dimension uDimension * vDimension. This set of points defines a mesh similar to the grid of an ElevationGrid, whereas the points do not have a uniform spacing. Depending on the weight values and the order, this hull is approximated by the resulting surface. The number of uDimension points define a polyline in u-direction followed by further u-polylines with the v-parameter in ascending order. The number of control points must be equal to or greater than the order. A closed B-Spline surface can be specified by repeating the limiting control points and by specifying a periodic knot vector

The control vertex corresponding to the control point P[i, j] on the control grid is:

P[i,j].x = controlPoints[i + (j × uDimension)].x
P[i,j].y = controlPoints[i + (j × uDimension)].y
P[i,j].z = controlPoints[i + (j × uDimension)].z
P[i,j].w = weight[i + (j × uDimension)]
where 0 <= i < uDimension and 0 <= j < vDimension.

A ***weight*** value that must be greater than zero is assigned to each controlPoint. The ordering of the values is equivalent to the ordering of the control point values. If the weight of a control point increased above 1 the point is more closely approximated by the surface. However the surface is not changed if all weights are multiplied by a common factor. The number of values must be identical to the number of control points. If the length of the weight vector is 0, the default weight 1.0 is assumed for each control point.

As a result of the lack of a 4D Coordinate field type in VRML, the control points and the corresponding weight values are held in separate fields. This separation also allows independent animation of the controlPoint fields using a CoordinateInterpolator node.
uKnots and ***vKnots*** define the knot vector. The number of knots must be equal to the number of control points plus the order of the curve. The order must be non-decreasing. By setting successive knot values equal, the degree of continuity is decreased, which implies that the surface gets edges. In general, the curve or surface is of continuity C^{k-1-m} at a knot point, where k is the order and m is the number of consecutive knots being equal. If k is the order of the curve, k consecutive knots at the end or the beginning of the vector cause the curve to interpolate the last or the first control point respectively. Within the knot vector there may not be more than k-1 consecutive knots of equal value. If the length of a knot vector is 0, a default uniform knot vector is computed.

uTessellation and ***vTessellation*** give hints to the surface tessellator, u/v Tessellation > = u/v Order sets an absolute number of subdivision steps, 0 lets the browser choose a suitable tessellation. Interpretation of values below 0 are implementation-dependent.

For an implementation subdividing the surface into an equal number of subdivision steps, tessellation values could be interpreted in the following way: if a tessellation value is greater than 0, the number of tessellation points is tessellation+1; if a tessellation value is smaller than 0, the number of tessellation points is (-tessellation * (u/v)dimension)+1 if a tessellation value is 0, the number of tessellation points is (2 * (u/v)dimension)+1
For implementations doing tessellations based on chord length, tessellation values <0 could be interpreted as the maximum chord length deviation in pixels. Implementations doing fully automatic tessellation may ingore the tessellation hint parameters.

texCoord could provide additional information on how to generate texture coordinates. By default, texture coordinates in the unit square are generated automatically from the parametric subdivision. It is under consideration to use a NurbsTextureSurface Node or simply a TextureCoordinate node in order to be able to compute a texture coordinate given a u/v parameter of the NurbsSurface. NurbsTextureSurface would also allow for non-animated surfaces to specify a chord-length-based texture coordinate parametrization. Feedback from content developers is required to resolve this open issue.

ccw and ***solid*** are defined like in other VRML Geometry nodes. solid TRUE enables two-sided lighting, the surface is visible from both sides, and normals are flipped toward the viewer prior to shading.

NurbsCurve

```
NurbsCurve {
field         MFFloat knot          []    # (-∞,∞)
field         SFInt32 order         3     # [2, ∞)
exposedField MFVec3f controlPoint  []    # (-∞,∞)
exposedField MFFloat weight        []    # (0, ∞)
exposedField SFInt32 tessellation  0     # (-∞,∞)
}
```

The NurbsCurve node is defined analogous to the NurbsSurface node. The dimension field is left out, since these fields are only necessary to resolve the set of the surface control points as a 2 dimensional area.

NurbsGroup

```
NurbsGroup {
eventIn      MFNode  addChildren
eventIn      MFNode  removeChildren
exposedField MFNode  children   []
field        SFVec3f bboxCenter 0 0 0    # (-∞,∞)
field        SFVec3f bboxSize   -1 -1 -1 # (0,∞)
or -1,-1,-1
exposedField SFFloat tessellationScale 1.0
}
```

The NurbsGroup node groups a set of NurbsSurface nodes to a common group. This provides a hint to the browser to treat the set of NurbsSurface as a unit during tessellation to enforce tessellation continuity along borders. The **tessellationScale** parameter scales the tessellation values in lower-level NurbsSurface nodes. If a set of NurbsSurfaces uses a matching set of controlPoints along the borders, this results in a common tessellation stepping.

NurbsPositionInterpolator

```
NurbsPositionInterpolator {

  eventIn      SFFloat set_fraction
  exposedField SFBool   fractionAbsolute TRUE
  exposedField SFInt32  dimension        0
  exposedField MFFloat  knot             []
  exposedField SFInt32  order            4
  exposedField MFVec3f  keyValue         []
  exposedField MFFloat  keyWeight        []
  eventOut     SFVec3f  value_changed
}
```

NurbsPositionInterpolator describes a 3D NURBS Curve using *dimension keyValue*, *keyWeight*, *knot* and *order*. Sending a *set_fraction* input computes a 3D position on the curve, which is sent by *value_changed*. The *set_fraction* value is used as the input value for the tessellation function. Thereby the *knot* coresponds to the *key* field of a conventional interpolator node, i.e. if the *set_fraction* value is within [0;1] and the knot vector within [0;2] only the half of the curve is computed. To traverse an arbitrary knot span with the normal input span of [0;1] of a TimeSensor node a mapping function can be activated by setting the *fractionAbsolute* to FALSE.

It is under consideration to expand the functionality to also compute tangents:

```
exposedField SFBool     computeTangent FALSE
eventOut     SFVec3f    tangent_changed
```

The tangent is computed if the *computeTangent* field is TRUE. This eventOut can be used to compute a frame of reference at the current position along the curve.

Using a VRML PositionInterpolator, it is not possible to specify a smooth movement like a path along a circle until the curve is sampled to a very fine resolution. In a lot of existing VRML content, the data for Interpolators make up a substantial portion of the total size of the VRML file. Using Spline- (NURBS-) based interpolation, we hope that this amount of data can be reduced. Feedback from content developers and tool vendors is required to evaluate this node and its usability and data reduction capabilities.

NurbsTextureSurface

```
NurbsTextureSurface {
field        SFInt32  uDimension   0     # [0, ∞)
field        SFInt32  vDimension   0     # [0, ∞)
field        MFFloat  uKnot       []     # (-∞,∞)
field        MFFloat  vKnot       []     # (-∞,∞)
field        SFInt32  uOrder       3      # [2, ∞)
field        SFInt32  vOrder       3      # [2, ∞)
exposedField MFVec2f  controlPoint []     # (-∞,∞)
exposedField MFFloat  weight       []     # (0, ∞)
}
```

The NurbsTextureSurface node is a NURBS surface existing in the parametric domain of its surface host specifying the mapping of the texture onto the surface. The tessellation process generates 2D texture coordinates, which means additional computational effort in a frame-by-frame tessellation. If the NurbsTextureSurface is undefined, texture coordinates are computed by the client on the basis of parametric step size without any performance penalty. Conventional vertex parameters do not apply on NURBS because triangles are only available after polygonization, but the conventional texture transform may be used.

TrimmedSurface

```
TrimmedSurface {
eventIn      MFNode  addTrimmingContour
eventIn      MFNode  removeTrimmingContour
exposedField MFNode  trimmingContour   []
exposedField SFNode  surface          NULL
}
```

The TrimmedSurface node defines a NURBS surface that is trimmed by a set of trimming loops. The **surface** field contains the **NurbsSurface** that shall be trimmed. The **trimmingContour** field, if specified, shall contain a set of **Contour2D** nodes. The contours specify the area to trim out following the trimming rule aligned to the Open Inventor definition [Wer94]. A area inside a loop is discarded if the loop is defined in a clockwise direction. If the loop is defined in a counterclockwise direction the area inside is retained and outside is discarded. The outermost contour must be defined in a counterclockwise direction. The contours may not be self-intersecting or intersect other contours.

Contour2D

```
Contour2D {
eventIn      MFNode  addChildren
eventIn      MFNode  removeChildren
exposedField MFNode  children   []
field        SFVec3f bboxCenter 0 0 0    # (-∞,∞)
field        SFVec3f bboxSize   -1 -1 -1 # (0, ∞)
or -1,-1,-1
}
```

The Contour2D node groups a set of curve segments to a composite contour. The children have to form a closed loop with the first point of the first child repeated as the last point of the last child and the last point of a segmet repeated as the first point of the consecutive one. The segments must be of the type

NurbsCurve2D or **Polyline2D** and must be enumerated in the child field in consecutive order according to the topology of the contour.

NurbsCurve2D

```
NurbsCurve2D  {
field        MFFloat knot            []
field        SFInt32 order           3
exposedField MFVec2f controlPoint    []
exposedField MFFloat weight          []
exposedField SFInt32 tessellation    0
}
```

The NurbsCurve2D node defines a trimming segment that is part of a trimming contour in the u-v domain of the surface. If the NurbsCurve2D forms a closed contour, it may be used as a Contour2D node.

Polyline2D

```
Polyline2D  {
exposedField MFVec2f  point    []
}
```

The Polyline2D node defines a linear curve segment as a part of a trimming contour in the u-v domain of a surface.

3. IMPLEMENTATION

Our implementation in the VRML browser blaxxun Contact [bla2, web] proves the usage of NURBS in the VRML domain. All proposed nodes except the NurbsTextureSurface have been successfully implemented. In favour of fast and stable processing a uniform tessellation was applied as recommended in [KML96].
Various rendering pipelines fullfill the requirements of different platforms (Figure 2). As a reference implementation we used the OpenGL tessellator located in the glu-library. The NURBS surface parameters are directly handed over to the corresponding OpenGL functions. This method is easy to implement, but performance is low.

To use the built in rendering pipline in our client, we polygonize NURBS models and store the resulting mesh data. If no dynamic tessellation is required, the performance penalty is avoided alltogether. For low-end CPUs or complex models, this method can be used to tessellate NURBS surfaces in a preprocessing step. In case of dynamic tessellation (view-depended rendering) the introduction of thresholds for the tessellation values minimizes the computational load. If the tessellation value is altered by more than 10% the vertex cache is flushed and a new tessellation cycle is invoked.

Current chip architectures like the Intel ISSE or the AMD 3DNow! allow to parallelize the computations involved in the polygonization. An ISSE optimized tessellation with additionally optimized lighting and transformation showed best results. This pipeline exploits the fact that CVs are invariant under transformations. Transform is sped up by applying the costly matrix multiplication to the small amount of CVs. After the tessellation process the transformed vertices are lit.

In the current implementation stage no step size computation is performed. As a first step we encourage the content author to set a suitable tessellation bound simply based on the desired smoothness. In addition, this value can be used to weight the polygon budget of objects. Essential objects are given a high tessellation value to ensure a smooth surface; less important objects get a lower tessellation value because some coarseness is tolerable.

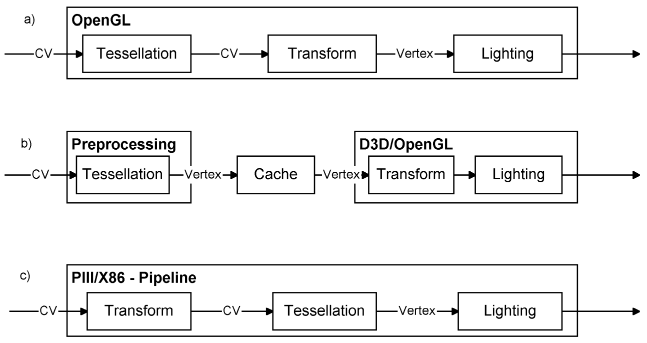

Figure 2: pipelines for tessellation and rendering

View dependent rendering

The implementation is targeted to dynamic tessellation which allows view-dependend rendering. Like the classical LOD, NURBS objects can be scaled down according to the viewer's distance from the object. We have introduced a quality factor describing the rendering quality by means of triangles per screen size of the object. This value is comparable with the screen based tolerance shown in section 1, because both values specify the resolution of an object. In contrast to the screen based tolerance the computation is very simple but is not directly related to the curvature of the object

$$Quality = \frac{triangles(scale)}{bbox(dist)}$$

With
$triangles(scale) = (uTess + 1) * scale * (vTess + 1) * scale * 2$

Quality:	Describes rendering quality of an object
triangles(scale):	Number of triangles of an object
bbox(dist):	Diameter of the bounding box in screen space

The quality factor is specified by the tessellation values in VRML. If the quality is assumed to be constant the scale value alters with the distance of the viewer to the object. An additional feedback loop can scale the quality factor according to the CPU speed reflecting in the frame rate. The algorithm keeps the framerate constant by scaling the NURBS content at any given CPU speed. Of course, this implies that the whole scene has to be designed for scaling: There has to be reasonable space for upscaling and downscaling the tessellation and thus the appearance of an object if moving away from the target platform. However, scalability is not unlimited: In the average scene NURBS content is scaled down very fast with CPU frequency since certain portions of the overall computational effort do not decrease. The drawbacks of CPU based scaling are that the authors lose control over the appearance of the object since the client itself continuously alters the object's resolution. Proper lower bounds have to be defined to avoid intolerable undersampling of the model. To overcome these shortcomings additional parameters could be introduced: Authors could specify the tessellation strategy or the lower boundary of the quality.

Other parametric shapes like cylinders, cones and spheres can profit from the NURBS LOD as well. A client could compose these shapes with NURBS “behind the scenes,“ allowing seamless scaling of the objects.

Large-scale NURBS surfaces are not addressed in this approach and have to be subdivided by hand. In the worst case, a huge surface is very coarse nearby while totally oversampled in its remote parts. To avoid boundary cracks of adjacent surfaces due to the view dependent rendering NURBS objects have to be grouped in a NurbsGroup node.

Trimmed NURBS

The usage of the NURBS tessellator of OpenGL for the presented nodes in the context of trimmed NURBS is straightforward. The Contour2D node signals OpenGL the beginning and the end of a trimming loop. The NurbsCurve2D coresponds to a gluNurbsCurve and the Polyline2D is the VRML counterpart of gluPwlCurve. The performance of glu's NURBS implementation on the average PC (table 1) is low. In the trimmed NURBS case performance decreased considerably in comparison to the untrimmed one, reserving the OpenGL implementation to high end workstations, at least until graphic boards supporting NURBS in hardware on basis of the OpenGL API arise on the market. Therefore in our implementation the performance penalty of tessellation was avoided by caching the tessellated model. To render even complex CAD models an optimal polygon count is achieved by employing adaptive tessellation. Techniques relying on realtime tessellation such as the animation of control vertices and view depended rendering are out of the scope in this implementation, but various LOD modells could be generated in a preprocessing step. Further work has to be done in using trimmed NURBS for LODs.

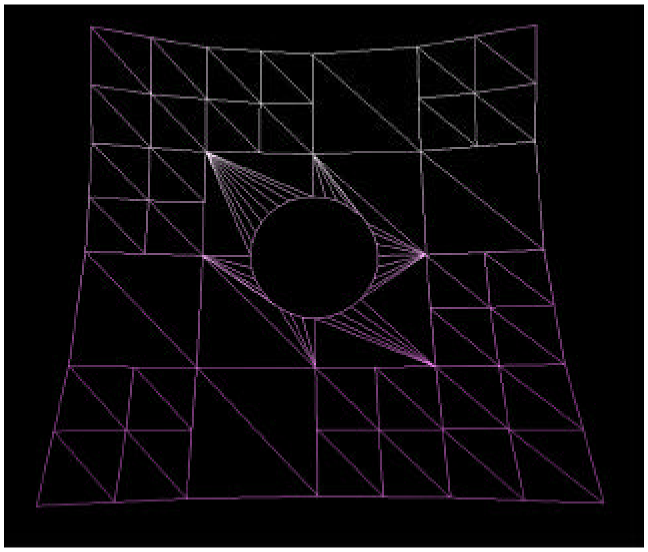

Figure 3 Triangulation of a trimmed NURBS patch by adaptive tessellation

4. Results

For testing we take as a common scenario a scene with multiple NURBS objects shown at high resolution and exploiting the advantage of arbitrary resolution for a LOD. Multiuser worlds with a huge number of avatars moving around or a shopping mall containing a large number of products show this behaviour.

A sample implementation [bla2] of file exporters for 3d Studio Max 2.5 and 3.0 respectively supports the new nodes. The NURBS export function is fully integrated in the conventional VRML97 file exporter in the 3.0 version. In another approach we convert Open Inventor files to VRML. This method is straightforward since the specification of the proposed nodes is closely related to the one of Open Inventor NURBS. Supporting authoring tools like 3d Studio Max seems to be more complex: Internal NURBS data structures like ruled, loft and blended surfaces have to be reduced to the basic NURBS surface. Various tools especially in the CAD domain support the Open Inventor file format. For professional modelling Alias Wavefront can be used.

IFS in D3D	OpenGL	Preprocessing	ISSE
60fps	5fps	38fps	45fps

Table 1: Comparison between tessellation pipelines. The scene contained an animated NURBS patch. Uniform tessellation was configured to 5000 triangles. The frame rate increases noticeable if the patch is stored as a IndexedFaceSet.

Animation with NURBS

The frame-by-frame tessellation allows us to alter the position of the control points on the fly. To achieve effects like moving waves with conventional meshes, many points have to be animated at the cost of huge data size of CoordinateInterpolators or complex scripts. By transferring this problem to NURBS, only single control points have to be animated. Data size is kept very small and scripts are very simple.

Currently this technique is only limited by the lack of suitable authoring tools. Awesome effects can be achieved by-hand editing the VRML file, but as the complexity of objects increases, tools have to be used. A sample implementation based on 3D Studio MAX shows the power of NURBS animations. The sampled animation path is approximated by a spline and exported as a NurbsPositionInterpolator.

Compression

The figures presented in this section can only give a idea of the compression factor, since the factor largly depends on the degree of curvature of the model. A NURBS model in 3d Studio Max was exported to the VRML97 NURBS format and to a conventional IndexedFaceSet with a resolution such that the model did not show any rendering artefacts. However, this comparison assumes that the NURBS model is shown at its optimal resolution.

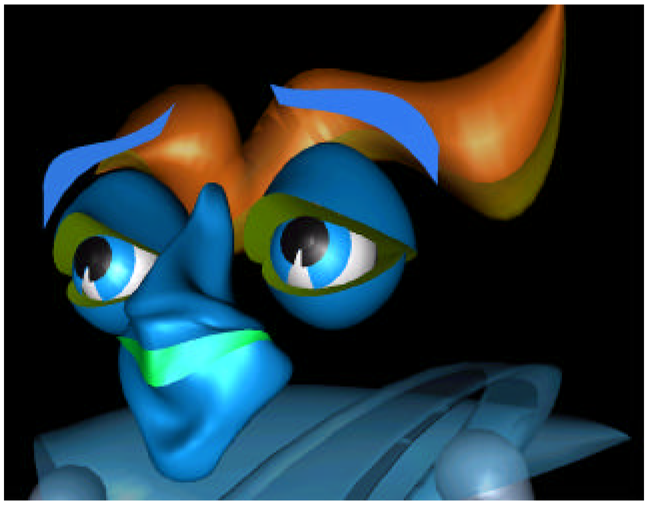

Figure 4 The Knight NURBS avatar composed of several NURBS patches. Even at close ups the avatar stays smooth. The avatar is taken out of the Knight NURBS demo of Lunatic Interactive (www.lunatic.de).

	IFS	NURBS
a) curved surface	66K (6.8K Polygons)	7K
b) Knight NURBS	2MB (40K Polygons)	200K

Table 2 File size in KByte. a) A tool generated sphere representing an average curved surface was exported as NURBS surface (without expoiting the special characteristics of a sphere) and as a mesh. b) A NURBS avatar shown in fig. 4.

Speed improvement with LOD

Depending on the scene, the LOD can speed up the rendering process tremendously. Especially in a multiuser sceneario where many avatars are moving in a scene, the LOD has a tremendous impact on the performance. In a typical multiuser scenario, the complexity of the environment is negligible in comparison to that of 50 or even 100 avatars. Typically, avatars are tool-generated and thus only poorly VRML-optimized. Experiments with NURBS avatars have shown acceleration of at least 2x. Authoring various LOD models for the conventional LOD node of VRML is a time-consuming process whereas with NURBS the LODs come for free.

	IFS	NURBS
fps	8	24
polygons	150K	35K

Table 3 Frame rate increases by factor 3 in this example when employing NURBS avatars with LOD instead of IndexedFaceSet based avatars. The overall polygon load of the scene decreased from 150K polygons to 35K since only the avatars close to the camera are of high resolution.

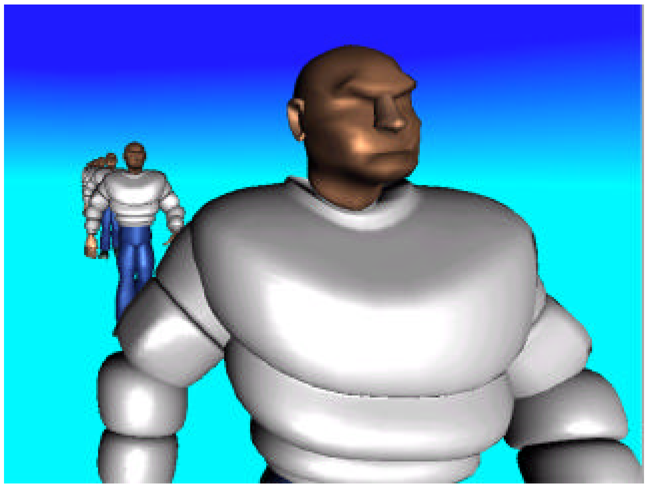

Figure 5 A scene with 10 high resolution avatars (15K Polygons) positioned in different distances to the viewer was rendered as IndexedFaceSet and as NURBS model with LOD. The avatar was provided by the courtesy of okupi (www.okupi.com).

5.SUBDIVISION SURFACES

Recently, subdivision surfaces have gained more attention as a possible alternatives to NURBS. In particular, in areas such as animation and entertainment, subdivision surfaces have been employed. The reason for their increasing popularity is the fact that it is possible to generate geometry with arbitrary topology without introducing trimming. This simplifies the user interaction considerably and also eliminates the robustness problems which are inherent to trimmed NURBS.

The basic idea of subdivision surfaces is the following: Given is a polygonal mesh M^0. A refined mesh M^1 is created by subdividing each face into a collection of subfaces. Typically, the vertices of the new mesh M^1 are computed as weighted averages of the vertices of the old mesh. Repeated application of this subdivision step leads to a smooth limit surface. The most popular schemes are Catmull-Clark subdivision [CC78] and Loop subdivision [Loo87]. The two schemes differ in that Catmull-Clark is based on rectangles and the limit surface is a bicubic B-Spline. Loop‘s scheme is based on triangles. For commercial applications, Catmull-Clark is favored since it directly generalizes the bicubic tensor product B-Splines.

This scheme has been used at Pixar for the creation of geometry in the short film Geri‘s game. In [DKT98] , the authors describe techniques developed by Pixar for modelling sharp creases and for texture mapping subdivision surfaces. Catmull-Clark subdivision surfaces are also implemented in Alias/Wavefront Maya.

Evaluation of Subdivision Surfaces

It was a long-held belief that there is no direct way of evaluating a subdivision surface efficiently. Looking at Figure 6 we can see that almost everywhere the subdivision surface consists of quadrilaterals which forms the control net of a bicubic B-Spline. Only around the *extraordinary* vertices (vertices of valence not equal 4) does the surface not correspond to a B-Spline. Stam [Sta98] showed that even around extraordinary vertices, it is possible to evaluate the surface directly and efficiently. This can be achieved by transforming to a suitable basis, the eigenbasis which correpsonds to the power basis in regular regions. Hence, it is now much more practical to incorporate subdivision surfaces into a modeling framework and to have them coexist with standard NURBS.

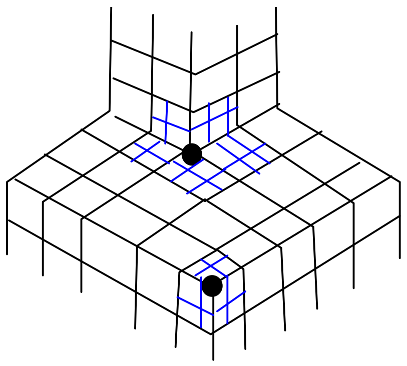

Figure 6 Subdivision mesh with extraordinary vertices. Only the marked patches can not be evaluated directly as bicubic B-Splines.

Level of Detail Control

One of the strengths of subdivision surfaces is the built-in multiresolution hierarchy. A mesh can be refined by performing additional subdivision steps. Conversely, it is also possible to coarsen the mesh by applying mesh decimation techniques. In [ZSS97], the authors describe techniques for adding detail locally, which allows to edit the model in a restricted region. It is also shown how one can traverse the multiresolution hierarchy in order to perform adaptive rendering. This enables the application to trade off frame rates for level of detail as described in Section 3.

Summary

It is possible to put forth a proposal for subdivision surfaces in the future which can make use of the NURBS standard proposed in this paper. Subdivision surfaces address some issue which are problematic for NURBS such as the modeling of arbitrary topology. On the other hand, NURBS are well-established in the engineering community and they are superior for modeling in an engineering context. It is difficult to compare the storage and hence the transmission requirements. A model which has lots of detail in a local region favors a subdivision surface since it allows to add detail locally, whereas in NURBS we have to modify the patch globally. On the other hand, storing a full multiresolution hierarchy can increase the transmission cost for subdivision surfaces, whereas LOD control comes essentially for free when rendering NURBS. In any case, both respresentations yield significant compression ratios when compared to tessellated models.

6. CONCLUSION

NURBS present a great opportunity to meet the requirements of current hardware and the Internet. With NURBS scalable web application can be created which react to the configuration of the client system. Practical experience has proven that NURBS are effective with current hardware and can save considerable bandwidth. Besides this obvious advantage of data compression, the LOD is a key feature. Defining various levels of detail of a model is not left up to the author but computed automatically by the application.

As a first step, work on the standardization of the proposed NURBS nodes needs to be done. Secondly, a standard for the tessellation and for the LOD technique have to be agreed upon to guarantee same visual appearance of objects in various implementations.

Further work has to do be done in the field of authoring solutions. Due to the complexity of NURBS, objects cannot be modeled by hand in a text editor. We have to develop file exporters and file converters for the majority of the authoring tools to increase the acceptance of NURBS in the community of content developers. Looking farther ahead, one can imagine to incorporate subdivision surfaces as well. Again, the issue of authoring solutions will have to be addressed.

REFERENCES

[AES91] S.S. Abi-Ezzi, L.A. Shirman. Tessellation of curved surfaces under highliy vaying transformations. Proceedings of Eurographics 91, pages 385-397, 1991

[AES94] S.S. Abi-Ezzi, S. Subramaniam, Fast Dynamic Tessellation of Trimmed NURBS Surfaces, Compter Graphics Forum, Vol. 13 (3), 1994

[bla1] http://www.blaxxun.com/developer/contact/3d/nurbs/overview.html

[bla2] blaxxun contact: http://www.blaxxun.de/download/contact/index.html

[Dah86] W. Dahmen: Subdivision algorithms converge quadratically, J. Comp. Appl. Math., Vol 16., pp. 125-158, 1986

[DKT98] T. DeRose, M. Kass, T. Truong, Subdivision Surfaces in Character Animation. Computer Graphics Proceedings (SIGGRAPH 98), pp 85-94, 1998

[Far96] G. Farin, Curves and Surfaces for Compter Aided Geometric Design: A Practical Guide, 4th Edition, Academic Press, Boston 1996

[FMM86] D. Filip, R. Magedson, R. Markot. Surface algorithms using bounds on derivatives. CAGD (3), 295-311, 1986

[KML96] S. Kumar, D. Manocha, A. Lastra, Interactive Display of large scale NURBS models, IEEE Transactions on visualization and Computer Graphics, Vol. 2, December, 1996

[LC93] W.L. Luken and Fuhua Cheng, Rendering Trimmed NURB Surfaces, Computer Science Research Report 18669(81711), IBM Research Division, 1993

[Luk93] W.L. Luken, Tessellation of Trimmed NURB Surfaces, Computer Science Research Report 19322(84059), IBM Research Division, 1993

[Loo87] C. Loop, Smooth Subdivision Surfaces Based on Triangles. Master‘s thesis, University of Utah, Department of Mathematics, 1987

[Pet94] J.W. Peterson, Tessellation of NURBS Surfaces, Graphic Gems IV, Academic Press, p.286-320, Boston, 1994

[PT95] L. Piegl, W. Tiller, The NURBS Book, Springer, Heidelberg, 1995

[RHD89] A. Rockwood, K. Heaton, T. Davis. Realtime rendering of trimmed surfaces. In Proceedings of ACM Siggraph, pages 107-117, 1989

[sgi] http://www.sgi.com/Technology/Inventor/VRML/VRMLDesign.html

[Sta98] J. Stam, Exact Evaluation of Catmull-Clark Subdivision Surfaces at Arbitrary Parameter Values. Computer Graphics Proceedings (SIGGRAPH 98) pp 395-404, 1998

[web] source code of blaxxun Contact: http://www.web3d.org/TaskGroups/source/blaxindex.html

[Wer94] J. Wernecke, The Inventor Mentor, Addison-Wesley, 1994

[ZSS97] D. Zorin, P. Schroeder, W. Sweldens, Interactive Multiresolution Mesh Editing. Computer Graphics Proceedings (SIGGRAPH 97) pp. 259-268, 1997